Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Амурский государственный университет»
(ФГБОУ ВПО «АмГУ»)

Кафедра ФИЗИКИ

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ДИСЦИПЛИНЫ

«ОРГАНИЗАЦИЯ БАЗ ДАННЫХ»

Основной образовательной программы по специальности 010701.65 – «Физика»

Благовещенск 2012

УМКД разработан кан. физ.-мат. наук, доцентом кафедры ФИЗИКИ Стуковой Еленой Владимировной

Рассмотрен и рекомендован на заседании кафедры ФИЗИКИ

Протокол заседания кафедры от «___»_____2012г. №_____

И.о. зав. кафедрой

_____/_____/

(подпись) (И.О. Фамилия)

УТВЕРЖДЕН

Протокол заседания УМСС 010701.65 – «Физика»
от «___»_____2012г. №____

Председатель
УМСС_____/_____/

(подпись) (И.О. Фамилия)

Содержание

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Амурский государственный университет»

УТВЕРЖДАЮ
Проректор по учебной работе
_____В.В. Проказин
«_____»_____ 201__г.

РАБОЧАЯ ПРОГРАММА
Организация баз данных
(наименование учебной дисциплины/модуля)

для специальности 010701.65 – «Физика»

по специализациям: «Информационные технологии в образовании и научной деятельности»
«Медицинская физика»

Квалификация выпускника: ФИЗИК

Курс IV

Семестр 8
Курсовая работа _8_
(семестр)

Экзамен _8_
(семестр)

Лекции 36 (час.)
Лабораторные работы 18 (час.)
Практические занятия 18 (час.)
Самостоятельная работа 13 (час.)
Общая трудоемкость дисциплины 85 (час.)

Составитель Е.В. Стукова, доцент, канд. физ.-мат. наук.
(И.О.Ф., должность, ученое звание)

Факультет: инженерно-физический
Кафедра: теоретической и экспериментальной физики

Благовещенск 2012 г.

Рабочая программа составлена на основании Государственного образовательного стандарта высшего профессионального образования и авторских разработок по направлению подготовки 010701.65 – «Физика», квалификация: физик.

Рабочая программа обсуждена на заседании кафедры теоретической и экспериментальной физики
«___»_____ 201__г., протокол № _____
И.о. заведующего кафедрой_____ Е.А.Ванина

Рабочая программа одобрена на заседании учебно-методического совета по направлению подготовки 010701.65 – «Физика»
«___»_____201__г., протокол №_____
Председатель УМСС_____

Рабочая программа утверждена на заседании кафедры теоретической и экспериментальной физики от _____
протокол № _____
Председатель_____ Е.А. Ванина

Рабочая программа переутверждена на заседании кафедры теоретической и экспериментальной физики от _____
протокол № _____
И.о. заведующего кафедрой_____ Е.А.Ванина

СОГЛАСОВАНО
Начальник УМУ
_____
«___»_____201__ г.

СОГЛАСОВАНО
Председатель УМС факультета
_____
«___»_____201__ г.

СОГЛАСОВАНО
Заведующий выпускающей кафедрой
_____
_____
«___»_____201__г.

СОГЛАСОВАНО
Директор научной библиотеки
«___»_____201__г.

1. ЦЕЛИ И ЗАДАЧИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Цель дисциплины заключается в ознакомлении студентов с основными принципами организации баз и банков данных, а также получении теоретических знаний и практических навыков по проектированию и разработке баз данных.

Задачами дисциплины являются приобретение знаний: об основных этапах проектирования баз данных; моделях данных (иерархической, сетевой и реляционной); принципах нормализации отношений; реляционной алгебре и реляционном исчислении; внутренней организации реляционной СУБД; ознакомлении с технологией “клиент-сервер”, современными промышленными СУБД и перспективами их развития.

2. МЕСТО ДИСЦИПЛИНЫ В СТРУКТУРЕ ООП ВПО:

Дисциплина «Организация баз данных» является дисциплиной, входящей в блок дисциплин специализаций СД.ДС.Р.14 для специальности 010701 «Физика».

Для освоения дисциплины необходимо знать:

1) информатику;
2) дискретную математику.

3. ЗНАНИЯ И УМЕНИЯ ОБУЧАЮЩЕГОСЯ, ФОРМИРУЕМЫЕ В РЕЗУЛЬТАТЕ ОСВОЕНИЯ ДИСЦИПЛИНЫ

В результате изучения дисциплины студент должен:

иметь представление об основных понятиях банков данных и знаний; основных компонентах банка данных; функциях администратора банка данных; о роли и месте банков данных в информационных системах, преимуществах централизованного управления данными; архитектуре банка данных;

знать основные понятия о системах управления базой данных (СУБД); инфологическое проектирование базы данных; выбор модели данных; иерархическая, сетевая и реляционная модели данных, их типы структур, основные операции и ограничения; представление структур данных в памяти

ЭВМ; современные тенденции построения файловых систем; основные типы промышленных СУБД; тенденции развития банков данных;

уметь проектировать и создавать базы данных на основе информационной модели предметной области, используя теоретические основы реляционных баз данных; выполнять запросы на изменение структуры базы, добавление, обновление и удаление данных, запросы на выборку и обработку данных на языке SQL; осуществлять основные функции по администрированию баз данных; создавать простейшие приложения баз данных;

иметь навыки работы в современных системах управления данными (MS-Access).

4. СТРУКТУРА И СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

Общая трудоемкость дисциплины составляет _85_ часов.

№ п/п	Раздел дисциплины	Виды учебной работы				Формы текущего контроля
		Лекции (час.)	Практические занятия (час.)	Лабораторные раб. (час.)	СРС (час.)	
1	ПОНЯТИЕ СУБД. ФУНКЦИИ СУБД. МОДЕЛИ БД	4	0	0	4	Подготовка разделов курсовой работы «Описание предметной области» и «Исследование информационных потребностей пользователей»
2	РЕЛЯЦИОННАЯ МОДЕЛЬ И ЕЕ ХАРАКТЕРИСТИКИ. ЦЕЛОСТНОСТЬ В РЕЛЯЦИОННОЙ МОДЕЛИ	2	2	4	-	Посещение лекций

3	РЕЛЯЦИОННАЯ АЛГЕБРА	4	2	0	-	Посещение лекций
4	ПРОЕКТИРОВАНИЕ БД. НОРМАЛЬНЫЕ ФОРМЫ ОТНОШЕНИЙ	6	2	6	6	Подготовка разделов курсовой работы «Инфологическое проектирование», «Логическое проектирование», «Физическое проектирование»
5	ПРОЕКТИРОВАНИЕ БД МЕТОДОМ СУЩНОСТЬ-СВЯЗЬ. ER-ДИАГРАММЫ	4	2	2	-	Письменный опрос
6	ЯЗЫК SQL	6	2	2	-	Посещение лекций
7	ОБЕСПЕЧЕНИЕ БЕЗОПАСНОСТИ БД	2	2	0	1	Посещение лекций
8	ФИЗИЧЕСКАЯ ОРГАНИЗАЦИЯ БД: СТРУКТУРЫ ХРАНЕНИЯ И МЕТОДЫ ДОСТУПА	2	2	0	-	Посещение лекций
9	ОПТИМИЗАЦИЯ ЗАПРОСОВ	4	2	2	-	Посещение лекций
10	ВОССТАНОВЛЕНИЕ ПОСЛЕ СБОЕВ	2	2	2	-	Посещение лекций
	Подготовка реферата по заданной теме				3	Реферат
	Итого в 8-м семестре	**36**	**18**	**18**	**13**	**Экзамен**

5. СОДЕРЖАНИЕ РАЗДЕЛОВ И ТЕМ ДИСЦИПЛИНЫ

5.1. Лекционный курс

ТЕМА I. ПОНЯТИЕ СУБД. ФУНКЦИИ СУБД. МОДЕЛИ БД

Понятие БД и СУБД. Уровни абстракции в СУБД. Функции абстрактных данных. Представления. Функции СУБД. Экспертные системы и базы знаний. Обзор ранних (дореляционных) СУБД. Системы, основанные на инвертированных списках. Иерархическая модель. Сетевая модель. Основные достоинства и недостатки ранних СУБД.

ТЕМА II. РЕЛЯЦИОННАЯ МОДЕЛЬ И ЕЕ ХАРАКТЕРИСТИКИ. ЦЕЛОСТНОСТЬ В РЕЛЯЦИОННОЙ МОДЕЛИ

Представление информации в реляционных БД. Домены. Отношения. Свойства и виды отношений. Целостность реляционных данных. Потенциальные и первичные ключи. Внешние ключи. Ссылочная целостность. Значения NULL и поддержка ссылочной целостности.

ТЕМА III. РЕЛЯЦИОННАЯ АЛГЕБРА

Понятие реляционной алгебры. Замкнутость в реляционной алгебре. Традиционные операции над множествами. Свойства основных операций реляционной алгебры. Специальные реляционные операции.

ТЕМА IV. ПРОЕКТИРОВАНИЕ БД. НОРМАЛЬНЫЕ ФОРМЫ ОТНОШЕНИЙ

Понятие проектирования БД. Функциональные зависимости. Тривиальные и нетривиальные зависимости. Замыкание множества зависимостей и правила вывода Армстронга. Неприводимое множество зависимостей. Нормальные формы – основные понятия. Декомпозиция без потерь и функциональные зависимости. Диаграммы функциональных зависимостей. Первая нормальная форма. Возможные недостатки отношения в 1НФ. Вторая нормальная форма. Возможные недостатки отношения во 2НФ. Третья нормальная форма. Возможные недостатки отношения в 3НФ. Нормальная форма Бойса-Кодда. Многозначные зависимости. Четвертая нормальная форма. Зависимости соединения. Пятая нормальная форма. Итоговая схема процедуры нормализации.

ТЕМА V. ПРОЕКТИРОВАНИЕ БД МЕТОДОМ СУЩНОСТЬ-СВЯЗЬ. ER-ДИАГРАММЫ

Возникновение семантического моделирования. Основные понятия метода. Диаграммы ER-экземпляров и ER-типа. Правила формирования отношений

.

ТЕМА VI. ЯЗЫК SQL

История создания и развития SQL.Основные понятия SQL. Запросы на чтение данных. Оператор SELECT. Итоговые запросы на чтение. Агрегатные функции. Запросы с группировкой. Вложенные запросы. Внесение изменений в базу данных.

ТЕМА VII. ОБЕСПЕЧЕНИЕ БЕЗОПАСНОСТИ БД

Общие положения. Методы обеспечения безопасности. Избирательное управление доступом. Обязательное управление доступом. Шифрование данных. Контрольный след выполняемых операций. Поддержка мер обеспечения безопасности в языке SQL. Представления и безопасность.

ТЕМА VIII. ФИЗИЧЕСКАЯ ОРГАНИЗАЦИЯ БД: СТРУКТУРЫ ХРАНЕНИЯ И МЕТОДЫ ДОСТУПА

Доступ к базе данных. Кластеризация. Индексирование. Структуры типа Б-дерева. Хеширование.

ТЕМА IX. ОПТИМИЗАЦИЯ ЗАПРОСОВ

Оптимизация в реляционных СУБД. Пример оптимизации реляционного выражения. Обзор процесса оптимизации. Преобразование выражений

ТЕМА X. ВОССТАНОВЛЕНИЕ ПОСЛЕ СБОЕВ

Понятие восстановления системы. Транзакции. Алгоритм восстановления после сбоя системы. Параллелизм. Проблемы параллелизма. Понятие блокировки. Решение проблем параллелизма. Тупиковые ситуации. Способность к упорядочению. Уровни изоляции транзакци.

5.2. Темы практических занятий

1. Проектирование конкретной БД (описание предметной области, определение границ предметной области, выявление информационных запросов пользователей). *4 часа*

2. Инфологическое проектирование (разработка спецификаций сущностей, атрибутов, связей, выбор ключей, создание справочника задач, построение концептуальной инфологической модели). *6 часов*

3. Логическое проектирование (проектирование реляционной логической модели базы данных, составление матрицы частоты совместного использования сущностей на основе справочника задач, оценка объема лишнего чтения, установление дополнительных логических связей, отображение инфологической модели на реляционную модель, получение совокупности отношений реляционной модели). *4 часа*

4. Нормализация отношений (приведение совокупности отношений к 1НФ, 2НФ, 3НФ).
2 часа
5. Физическое проектирование. *2 часа*

5.3. Темы лабораторных работ

1. Создание локальной базы данных, создание таблиц с помощью конструктора, целостность данных, создание ключей и индексов, определение типов данных. Маски ввода данных. *2 часа*
2. Схема базы данных, установление связей между таблицами. Обработка данных в таблицах. Сортировка и фильтрация данных. Обычные и расширенные фильтры. *2 часа*

3. Создание простых запросов, псевдонимы. *2 часа*

4.Создание запросов на основе нескольких таблиц. Выборка данных с условием. Использование выражений в запросах. *2 часа*

5. Соединение таблиц. Внутреннее, рекурсивное, внешнее левое и правое соединения, соединение по отношению. *2 часа*

6. Перекрестные запросы. Использование функций в запросах. *2 часа*

7. Запросы на создание, на обновление, на удаление, каскадное удаление и каскадное обновление данных. *2 часа*

8. Создание форм. Элементы управления формы. Диаграммы. Составные и связанные формы. Оформление формы. Ввод данных через форму. *2 часа*

9. Создание отчетов. Вычисляемые поля в отчете. Сортировка и группировка данных. *2 часа*

5.4. Курсовая работа

В качестве курсовой работы по дисциплине «Организация баз данных» студенты разрабатывают и реализовывают с помощью современной СУБД базу данных в предложенной им предметной области. Основные разделы, которые должны быть отражены в курсовой работе:

- описание предметной области;
- исследование информационных потребностей пользователей;
- инфологическое проектирование;
- логическое проектирование;
- физическое проектирование (реализация в СУБД по выбору);
- программный продукт;
- руководство пользователя;
- аппаратные и программные требования.

Темы курсовых работ:

1. Паспортный стол
2. Автомобильный магазин
3. Библиотека
4. Отдел сбыта и маркетинга ОАО Кондитерская фабрика "Зея"
5. Учет преступников
6. Деканат
7. Отдел налоговой полиции
8. Школа
9. Отдел кадров
10. Учет административных нарушений
11. Факультет дистанционного обучения
12. Музыкальный магазин
13. Регистрация транспортных средств
14. Детский сад
15. Отдел управления фирмы "Фармация"
16. Сведения об абитуриентах
17. Складской учет
18. Фирма по продаже компьютерного оборудования
19. Отдел аспирантуры и докторантуры
20. Поликлиника
21. Страхование
22. Станция технического обслуживания "Амур-Лада"
23. Гостиница
24. Ресторан

6. САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ

В течение семестра студентами должны быть подготовлен реферат по заданной теме и курсовая работа:

№ п/п	Форма самостоятельной работы	Кол-во уч. часов
1	Написание курсовой работы	10
2	Подготовка реферата на одну из заданных тем: 1. Функции, архитектура распределенных БД. 2. Преимущества и недостатки распределенных БД. 3. Фундаментальный принцип, свойства распределенных БД. 4. Технология клиент-сервер. 5. Связь объектно-ориентированных СУБД с общими понятиями объектно-ориентированного подхода. 6. Объектно-ориентированные модели данных. 7. Характеристики, достоинства и недостатки объектно-ориентированных СУБД. 8. Языки программирования объектно-ориентированных СУБД. 9. Языки запросов объектно-ориентированных СУБД. 10. Манифесты БД. 11. Характеристики объектно-реляционных СУБД. 12. Достоинства и недостатки объектно-реляционных СУБД. 13. Сравнительная характеристика объектно-ориентированных и объектно-реляционных СУБД. 14. Требования, предъявляемые к интеграции СУБД в среду Web. 15. Архитектура Web-СУБД. 16. Преимущества и недостатки интеграции СУБД в Web. 17. Основные методы интеграции СУБД в среду Web. 18. Безопасность Web-СУБД.	3
	Всего	13

7. ОБРАЗОВАТЕЛЬНЫЕ ТЕХНОЛОГИИ

Вид инноваций	Перечень инноваций
1. Методы, применяемые в обучении	Неимитационные методы обучения: *проблемная лекция, лекция-консультация.* Неигровые имитационные методы обучения: *контекстное обучение, метод решения творческих задач* (применяется в ходе практических занятий); *кейс-метод* (используется в ходе лабораторных занятий). Игровые имитационные методы: *мозговой штурм (*применяется на практических занятиях и на этапе защиты лабораторных работ)
2. Технологии обучения	Компетентностно-ориентированное обучение
3. Информационные технологии	Лекции проводятся с использованием интерактивной доски и мультимедийного оборудования.
4. Информационные системы	Электронный ресурс библиотеки АмГУ: http://www.biblio@amursu.ru/**.**
5. Инновационные методы контроля	Компьютерное интернет-тестирование.

8. ОЦЕНОЧНЫЕ СРЕДСТВА ДЛЯ ТЕКУЩЕГО КОНТРОЛЯ УСПЕВАЕМОСТИ, ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ ПО ИТОГАМ ОСВОЕНИЯ ДИСЦИПЛИНЫ И УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

Текущий контроль за аудиторной и самостоятельной работой обучаемых осуществляется во время проведения аудиторных занятий посредством устного опроса, проведения контрольных работ или осуществления лекции в форме диалога.

Промежуточный контроль осуществляется два раза в семестр в виде тестого задания (1-ая контрольная точка) в виде анализа разработанных схем по инфологическому и датологическому проектированию (2-я контрольная точка).

Экзамен – итоговый контроль осуществляется после успешного прохождения студентами текущего и промежуточного контроля в виде устного или письменного экзамена при ответах экзаменуемого на два вопроса в билете и дополнительные вопросы по желанию экзаменатора.

Студент, сдающий экзамен по данному предмету, должен показать знания по архитектуре и функциям БД, знать последовательность и содержание этапов проектирования БД, основные модели данных и конструкции языков манипулирования данными, разбираться в современном программном обеспечении систем управления базами данных.

Знания студента оцениваются как **отличные** при полном изложении теоретического материала экзаменационного билета и ответах на дополнительные вопросы со свободной ориентацией в материале и других литературных источниках.

Оценка "**хорошо**" ставится при твердых знаниях студентом всех разделов курса, но в пределах конспекта лекций и обязательных заданий по самостоятельной работе с литературой.

Оценку «**удовлетворительно**» студент получает, если дает неполные ответы на теоретические вопросы билета, показывая поверхностное знание учебного материала, владение основными понятиями и терминологией; при неверном ответе на билет ответы на наводящие вопрос.

Оценка «**неудовлетворительно**» выставляется за незнание студентом одного из разделов курса. Студент не дает полные ответы на теоретические вопросы билета, показывая лишь фрагментарное знание учебного материала, незнание основных понятий и терминологии; наводящие вопросы остаются без ответа.

Для допуска к экзамену студент должен сдать все лабораторные работы и реферат по самостоятельной работе.

8.1. Примерные вопросы к экзамену

1. Понятие БД и СУБД (Данные, аппаратное обеспечение, программное обеспечение, пользователи). Уровни абстракции в СУБД. Функции абстрактных данных.
2. Представления (внешенее, концептуальное, внутреннее).
3. Функции СУБД. Экспертные системы и базы знаний
4. Ранние (дореляционные) СУБД. Системы, основанные на инвертированных списках (структуры данных, манипулирование данными, ограничения целостности)
5. Иерархическая модель
6. Сетевая модель
7. Представление информации в реляционных БД
8. Домены. Отношения. Свойства и виды отношений
9. Целостность реляционных данных. Потенциальные и первичные ключи. Внешние ключи.
10. Ссылочная целостность. Правила внешних ключей. Значения NULL и поддержка ссылочной целостности
11. Понятие реляционной алгебры. Замкнутость в реляционной алгебре. Традиционные операции над множествами (объединение, пересечение, вычитание, произведение). Свойства основных операций реляционной алгебры
12. Специальные реляционные операции (выборка, проекция, естественное соединение, Θ-соединение, деление)
13. Специальные реляционные операции (операция расширения, операция подведения итогов, операторы обновления, реляционные сравнения)
14. Понятие проектирования БД
15. Функциональные зависимости. Тривиальные и нетривиальные зависимости

16. Замыкание множества зависимостей и правила вывода Армстронга
17. Неприводимое множество зависимостей
18. Декомпозиция без потерь и функциональные зависимости. Диаграммы функциональных зависимостей
19. Первая нормальная форма. Возможные недостатки отношения в 1НФ
20. Вторая нормальная форма. Возможные недостатки отношения во 2НФ
21. Третья нормальная форма. Возможные недостатки отношения в 3НФ. Сохранение зависимости
22. Нормальная форма Бойса-Кодда
23. Многозначные зависимости
24. Четвертая нормальная форма Зависимости соединения
25. Пятая нормальная форма Зависимости соединения, подразумеваемой потенциальными ключами Итоговая схема процедуры нормализации
26. Возникновение семантического моделирования. Основные понятия метода
27. Диаграммы ER-экземпляров и ER-типа. Связи типа 1:1, обязательный и необязательный классы принадлежности.
28. Диаграммы ER-экземпляров и ER-типа. Связи типа 1:M вариант H-O. Связи типа M:M и вариант класса принадлежности O-H
29. Правила формирования отношений. Степень связи 1:1(класс принадлежности обеих сущностей обязательный; класс принадлежности одной сущности обязательный, а второй – необязательный; класс принадлежности обеих сущностей – необязательный)
30. Правила формирования отношений. Степень связи между сущностями 1:M (или M:1) (класс принадлежности M-связной сущности обязательный; класс принадлежности M-связной сущности – необязательный). Степень связи M:M, независимо от класса принадлежности сущностей
31. Функции, архитектура распределенных БД.

32. Преимущества и недостатки распределенных БД.
33. Фундаментальный принцип, свойства распределенных БД.
34. Технология клиент-сервер.
35. Связь объектно-ориентированных СУБД с общими понятиями объектно-ориентированного подхода.
36. Объектно-ориентированные модели данных.
37. Характеристики, достоинства и недостатки объектно-ориентированных СУБД.
38. Языки программирования объектно-ориентированных СУБД.
39. Языки запросов объектно-ориентированных СУБД.
40. Манифесты БД.
41. Характеристики объектно-реляционных СУБД.
42. Достоинства и недостатки объектно-реляционных СУБД.
43. Сравнительная характеристика объектно-ориентированных и объектно-реляционных СУБД.

8.2. Пример теста для промежуточного контроля

1. База данных – это?
   1. набор данных, собранных на одной дискете;
   2. данные, предназначенные для работы программы;
   3. совокупность взаимосвязанных данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и обработки данных;
   4. данные, пересылаемые по коммуникационным сетям.

2. Иерархическая база данных – это?
   1. БД, в которой информация организована в виде прямоугольных таблиц;

2. БД, в которой элементы в записи упорядочены, т.е. один элемент считается главным, остальные подчиненными;
3. БД, в которой записи расположены в произвольном порядке;
4. БД, в которой существует возможность устанавливать дополнительно к вертикальным иерархическим связям горизонтальные связи.

3. Реляционная база данных - это?
   1. БД, в которой информация организована в виде прямоугольных таблиц;
   2. БД, в которой элементы в записи упорядочены, т.е. один элемент считается главным, остальные подчиненными;
   3. БД, в которой записи расположены в произвольном порядке;
   4. БД, в которой принята свободная связь между элементами разных уровней.

4. Сетевая база данных – это?
   1. БД, в которой информация организована в виде прямоугольных таблиц
   2. БД, в которой элементы в записи упорядочены, т.е. один элемент считается главным, остальные подчиненными;
   3. БД, в которой записи расположены в произвольном порядке;
   4. БД, в которой принята свободная связь между элементами разных уровней.
5. Поле – это?
   1. Строка таблицы;
   2. Столбец таблицы;
   3. Совокупность однотипных данных;
   4. Некоторый показатель, который характеризует числовым, текстовым или иным значением.

6. Запись – это?

1. Строка таблицы;
2. Столбец таблицы;
3. Совокупность однотипных данных;
4. Некоторый показатель, который характеризует числовым, текстовым или иным значением.

7. Характеристики типов данных. Убери лишнее.

1. Текстовый;
2. Поле МЕМО;
3. Числовой;
4. Функциональный;
5. Дата/число;
6) денежный;
7) словесный;
8) дата/время;
9) поле NEMO;
10) счетчик.

8. Форма – это?

1. Созданный пользователем графический интерфейс для ввода данных в базу;
2. Созданная таблица ввода данных в базу;
3. Результат работы с базой данных;
4. Созданная пользователем таблица.

9. Мастер – это?

1. Программный модуль для вывода операций;
2. Программный модуль для выполнения, каких либо операций;
3. Режим, в котором осуществляется построение таблицы или формы;
4. Режим, в котором осуществляется вывод таблицы или формы.

10. Конструктор – это?

1. Программный модуль для вывода операций;
2. Программный модуль для выполнения, каких либо операций;
3. Режим, в котором осуществляется построение таблицы или формы;
4. Режим, в котором осуществляется вывод таблицы или формы.

11. Виды работ с базами данных. Убери лишнее.
    1. Создание баз данных;
    2. Поиск данных;
    3. Сортировка данных;
    4. Заполнение базы данных;
    5. Создание формы данных;
    6. Отбор данных.

12. Какая панель используется для создания кнопки в базе данных?
    1. Инструментов;
    2. Компонентов;
    3. Элементов;
    4. Состояния.

9. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ «ОРГАНИЗАЦИЯ БАЗ ДАННЫХ»

а) основная литература:

1. **Илюшечкин, В. М**. Основы использования и проектирования баз данных [Текст] : учеб. пособие: рек. УМО / В.М. Илюшечкин. - М. : Высшее образование, 2009, 2011. - 214 с.
2. **Григорьев, Ю. А.** Теория и практика проектирования систем на основе баз данных [Текст] : учеб. пособие: рек. УМО / Ю. А. Григорьев, А. Д.

Плутенко. - Благовещенск: Изд-во Амур. гос. ун-та ; М.: Изд-во МГТУ им. Н.Э. Баумана, 2008. - 395 с.

3. **Астахова, И. Ф.** СУБД: язык SQL в примерах и задачах [Текст] : учеб. пособие: рек. Мин. обр. РФ / И. Ф. Астахова [и др.]. - М.: Физматлит, 2007. - 166 с.

б) дополнительная литература:

1. **Вейскас, Д.** Эффективная работа с Microsoft Access 2003 [Текст]: учебный курс / Д. Вейскас. – СПб.: Питер, 2001. – 398 с.
2. **Робинсон, С**. Microsoft Access 2000 [Текст]: учебный курс / С. Робинсон – СПб.: Питер, 2001. – 476 с.

в) периодическая литература:

- Журнал «КомпьютерПресс»;
- Журнал «Открытые системы. СУБД».

г) программное обеспечение и Интернет-ресурсы:

№	Наименование ресурса	Краткая характеристика
1	Сайт бесплатных электронных книг www.pitbooks.ru/subd	Сайт содержит необходимую литературу (Книги) и другой теоретический материал для самостоятельной работы студентов по Системам управления базами данных
2	Свободная энциклопедия Википедия http://ru.wikipedia.	Интернет-энциклопедия образовательных изданий, в которой собраны электронные учебники, справочники, а так же статьи различной тематики. Удобный поиск по ключевым словам, отдельным темам и отраслям знания.
3	Электронная библиотечная система **«Университетская библиотека- online»** www.biblioclub.ru	ЭБС по тематике охватывает всю область естественно-научных знаний и предназначена для использования в процессе обучения в высшей школе, как студентами, так и преподавателями.

10. МЕТОДИЧЕСКИЕ УКАЗАНИЯ К НАПИСАНИЮ КУРСОВОЙ РАБОТЫ

На первом этапе написание курсовой работы студенту необходимо выбрать тему (темы указаны в п.5.4). Тема может быть выбрана из предлагаемого перечня тем курсовый

МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Комплект ТСО

1. Интерактивная доска
2. Видеопроектор Epson
3. Мультимедийный проектор-03г
4. Ноутбук Пентиум 100-03г.

Программа составлена в соответствии с требованиями ГОС ВПО с учетом рекомендаций и ПрООП ВПО по специальности.

Тема 1. ПОНЯТИЕ СУБД. ФУНКЦИИ СУБД. МОДЕЛИ БД

С самого начала развития вычислительной техники образовались два основных направления ее использования. Первое направление - применение вычислительной техники для выполнения численных расчетов, которые слишком долго или вообще невозможно производить вручную. Становление этого направления способствовало интенсификации методов численного решения сложных математических задач, развитию класса языков программирования, ориентированных на удобную запись численных алгоритмов, становлению обратной связи с разработчиками новых архитектур ЭВМ.

Второе направление, которое непосредственно касается темы нашего курса, это использование средств вычислительной техники в автоматических или автоматизированных информационных системах. В самом широком смысле информационная система представляет собой программный комплекс, функции которого состоят в поддержке надежного хранения информации в памяти компьютера, выполнении специфических для данного приложения преобразований информации и/или вычислений, предоставлении пользователям удобного и легко осваиваемого интерфейса. Обычно объемы информации, с которыми приходится иметь дело таким системам, достаточно велики, а сама информация имеет достаточно сложную структуру. Классическими примерами информационных систем являются банковские системы, системы резервирования авиационных или железнодорожных билетов, мест в гостиницах и т.д.

На самом деле, второе направление возникло несколько позже первого. Это связано с тем, что на заре вычислительной техники компьютеры обладали ограниченными возможностями в части памяти. Понятно, что можно говорить о надежном и долговременном хранении информации только при наличии запоминающих устройств, сохраняющих информацию после выключения

электрического питания. Оперативная память этим свойством обычно не обладает. В начале использовались два вида устройств внешней памяти: магнитные ленты и барабаны. При этом емкость магнитных лент была достаточно велика, но по своей физической природе они обеспечивали последовательный доступ к данным. Магнитные же барабаны (они больше всего похожи на современные магнитные диски с фиксированными головками) давали возможность произвольного доступа к данными, но были ограниченного размера.

Легко видеть, что указанные ограничения не очень существенны для чисто численных расчетов. Даже если программа должна обработать (или произвести) большой объем информации, при программировании можно продумать расположение этой информации во внешней памяти, чтобы программа работала как можно быстрее.

С другой стороны, для информационных систем, в которых потребность в текущих данных определяется пользователем, наличие только магнитных лент и барабанов неудовлетворительно. Представьте себе покупателя билета, который стоя у кассы должен дождаться полной перемотки магнитной ленты. Одним из естественных требований к таким системам является средняя быстрота выполнения операций.

Как кажется, именно требования к вычислительной технике со стороны нечисленных приложений вызвали появление съемных магнитных дисков с подвижными головками, что явилось революцией в истории вычислительной техники. Эти устройства внешней памяти обладали существенно большей емкостью, чем магнитные барабаны, обеспечивали удовлетворительную скорость доступа к данным в режиме произвольной выборки, а возможность смены дискового пакета на устройстве позволяла иметь практически неограниченный архив данных.

С появлением магнитных дисков началась история систем управления данными во внешней памяти. До этого каждая прикладная программа, которой требовалось хранить данные во внешней памяти, сама определяла расположение каждой порции данных на магнитной ленте или барабане и выполняла обмены между оперативной и внешней памятью с помощью программно-аппаратных средств низкого уровня (машинных команд или вызовов соответствующих программ операционной системы). Такой режим работы не позволяет или очень затрудняет поддержание на одном внешнем носителе нескольких архивов долговременно хранимой информации. Кроме того, каждой прикладной программе приходилось решать проблемы именования частей данных и структуризации данных во внешней памяти.

Более точно, к числу функций СУБД принято относить следующие:

Непосредственное управление данными во внешней памяти

Эта функция включает обеспечение необходимых структур внешней памяти как для хранения данных, непосредственно входящих в БД, так и для служебных целей, например, для убыстрения доступа к данным в некоторых случаях (обычно для этого используются индексы). В некоторых реализациях СУБД активно используются возможности существующих файловых систем, в других работа производится вплоть до уровня устройств внешней памяти. Но подчеркнем, что в развитых СУБД пользователи в любом случае не обязаны знать, использует ли СУБД файловую систему, и если использует, то как организованы файлы. В частности, СУБД поддерживает собственную систему именования объектов БД.

Управление буферами оперативной памяти

СУБД обычно работают с БД значительного размера; по крайней мере этот размер обычно существенно больше доступного объема оперативной памяти. Понятно, что если при обращении к любому элементу данных будет

производиться обмен с внешней памятью, то вся система будет работать со скоростью устройства внешней памяти. Практически единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти. При этом, даже если операционная система производит общесистемную буферизацию (как в случае ОС UNIX), этого недостаточно для целей СУБД, которая располагает гораздо большей информацией о полезности буферизации той или иной части БД. Поэтому в развитых СУБД поддерживается собственный набор буферов оперативной памяти с собственной дисциплиной замены буферов.

Заметим, что существует отдельное направление СУБД, которое ориентировано на постоянное присутствие в оперативной памяти всей БД. Это направление основывается на предположении, что в будущем объем оперативной памяти компьютеров будет настолько велик, что позволит не беспокоиться о буферизации. Пока эти работы находятся в стадии исследований.

Управление транзакциями

Транзакция - это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется, и СУБД фиксирует (COMMIT) изменения БД, произведенные этой транзакцией, во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД. Понятие транзакции необходимо для поддержания логической целостности БД. Если вспомнить наш пример информационной системы с файлами СОТРУДНИКИ и ОТДЕЛЫ, то единственным способом не нарушить целостность БД при выполнении операции приема на работу нового сотрудника является объединение элементарных операций над файлами СОТРУДНИКИ и ОТДЕЛЫ в одну транзакцию. Таким образом, поддержание механизма транзакций является обязательным условием даже однопользовательских СУБД

(если, конечно, такая система заслуживает названия СУБД). Но понятие транзакции гораздо более важно в многопользовательских СУБД.

То свойство, что каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения, делает очень удобным использование понятия транзакции как единицы активности пользователя по отношению к БД. При соответствующем управлении параллельно выполняющимися транзакциями со стороны СУБД каждый из пользователей может в принципе ощущать себя единственным пользователем СУБД (на самом деле, это несколько идеализированное представление, поскольку в некоторых случаях пользователи многопользовательских СУБД могут ощутить присутствие своих коллег).

С управлением транзакциями в многопользовательской СУБД связаны важные понятия *сериализации транзакций* и *сериального плана выполнения смеси транзакций*. Под сериализаций параллельно выполняющихся транзакций понимается такой порядок планирования их работы, при котором суммарный эффект смеси транзакций эквивалентен эффекту их некоторого последовательного выполнения. Сериальный план выполнения смеси транзакций - это такой план, который приводит к сериализации транзакций. Понятно, что если удается добиться действительно сериального выполнения смеси транзакций, то для каждого пользователя, по инициативе которого образована транзакция, присутствие других транзакций будет незаметно (если не считать некоторого замедления работы по сравнению с однопользовательским режимом).

Существует несколько базовых алгоритмов сериализации транзакций. В централизованных СУБД наиболее распространены алгоритмы, основанные на синхронизационных захватах объектов БД. При использовании любого алгоритма сериализации возможны ситуации конфликтов между двумя или

более транзакциями по доступу к объектам БД. В этом случае для поддержания сериализации необходимо выполнить откат (ликвидировать все изменения, произведенные в БД) одной или более транзакций. Это один из случаев, когда пользователь многопользовательской СУБД может реально (и достаточно неприятно) ощутить присутствие в системе транзакций других пользователей.

Журнализация

Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя. Обычно рассматриваются два возможных вида аппаратных сбоев: так называемые мягкие сбои, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания), и жесткие сбои, характеризуемые потерей информации на носителях внешней памяти. Примерами программных сбоев могут быть: аварийное завершение работы СУБД (по причине ошибки в программе или в результате некоторого аппаратного сбоя) или аварийное завершение пользовательской программы, в результате чего некоторая транзакция остается незавершенной. Первую ситуацию можно рассматривать как особый вид мягкого аппаратного сбоя; при возникновении последней требуется ликвидировать последствия только одной транзакции.

Понятно, что в любом случае для восстановления БД нужно располагать некоторой дополнительной информацией. Другими словами, поддержание надежности хранения данных в БД требует избыточности хранения данных, причем та часть данных, которая используется для восстановления, должна храниться особо надежно. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД.

Журнал - это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью (иногда поддерживаются две копии журнала, располагаемые на разных физических дисках), в которую поступают записи обо всех изменениях основной части БД. В разных СУБД изменения БД журнализуются на разных уровнях: иногда запись в журнале соответствует некоторой логической операции изменения БД (например, операции удаления строки из таблицы реляционной БД), иногда - минимальной внутренней операции модификации страницы внешней памяти; в некоторых системах одновременно используются оба подхода.

Во всех случаях придерживаются стратегии "упреждающей" записи в журнал (так называемого протокола Write Ahead Log - WAL). Грубо говоря, эта стратегия заключается в том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем измененный объект попадет во внешнюю память основной части БД. Известно, что если в СУБД корректно соблюдается протокол WAL, то с помощью журнала можно решить все проблемы восстановления БД после любого сбоя.

Самая простая ситуация восстановления - индивидуальный откат транзакции. Строго говоря, для этого не требуется общесистемный журнал изменений БД. Достаточно для каждой транзакции поддерживать локальный журнал операций модификации БД, выполненных в этой транзакции, и производить откат транзакции путем выполнения обратных операций, следуя от конца локального журнала. В некоторых СУБД так и делают, но в большинстве систем локальные журналы не поддерживают, а индивидуальный откат транзакции выполняют по общесистемному журналу, для чего все записи от одной транзакции связывают обратным списком (от конца к началу).

При мягком сбое во внешней памяти основной части БД могут находиться объекты, модифицированные транзакциями, не закончившимися к моменту

сбоя, и могут отсутствовать объекты, модифицированные транзакциями, которые к моменту сбоя успешно завершились (по причине использования буферов оперативной памяти, содержимое которых при мягком сбое пропадает). При соблюдении протокола WAL во внешней памяти журнала должны гарантированно находиться записи, относящиеся к операциям модификации обоих видов объектов. Целью процесса восстановления после мягкого сбоя является состояние внешней памяти основной части БД, которое возникло бы при фиксации во внешней памяти изменений всех завершившихся транзакций и которое не содержало бы никаких следов незаконченных транзакций. Для того, чтобы этого добиться, сначала производят откат незавершенных транзакций (undo), а потом повторно воспроизводят (redo) те операции завершенных транзакций, результаты которых не отображены во внешней памяти. Этот процесс содержит много тонкостей, связанных с общей организацией управления буферами и журналом. Более подробно мы рассмотрим это в соответствующей лекции.

Для восстановления БД после жесткого сбоя используют журнал и архивную копию БД. Грубо говоря, архивная копия - это полная копия БД к моменту начала заполнения журнала (имеется много вариантов более гибкой трактовки смысла архивной копии). Конечно, для нормального восстановления БД после жесткого сбоя необходимо, чтобы журнал не пропал. Как уже отмечалось, к сохранности журнала во внешней памяти в СУБД предъявляются особо повышенные требования. Тогда восстановление БД состоит в том, что исходя из архивной копии по журналу воспроизводится работа всех транзакций, которые закончились к моменту сбоя. В принципе, можно даже воспроизвести работу незавершенных транзакций и продолжить их работу после завершения восстановления. Однако в реальных системах это обычно не делается, поскольку процесс восстановления после жесткого сбоя является достаточно длительным.

Поддержка языков БД

Для работы с базами данных используются специальные языки, в целом называемые *языками баз данных*. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков. Чаще всего выделялись два языка - *язык определения схемы* БД *(SDL - Schema Definition Language)* и *язык манипулирования данными (DML - Data Manipulation Language).* SDL служил главным образом для определения логической структуры БД, т.е. той структуры БД, какой она представляется пользователям. DML содержал набор операторов манипулирования данными, т.е. операторов, позволяющих заносить данные в БД, удалять, модифицировать или выбирать существующие данные. Мы рассмотрим более подробно языки ранних СУБД в следующей лекции.

В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных. Стандартным языком наиболее распространенных в настоящее время реляционных СУБД является язык SQL (Structured Query Language). В нескольких лекциях этого курса язык SQL будет рассматриваться достаточно подробно, а пока мы перечислим основные функции реляционной СУБД, поддерживаемые на "языковом" уровне (т.е. функции, поддерживаемые при реализации интерфейса SQL).

Прежде всего, язык SQL сочетает средства SDL и DML, т.е. позволяет определять схему реляционной БД и манипулировать данными. При этом именование объектов БД (для реляционной БД - именование таблиц и их столбцов) поддерживается на языковом уровне в том смысле, что компилятор языка SQL производит преобразование имен объектов в их внутренние идентификаторы на основании специально поддерживаемых служебных таблиц-

каталогов. Внутренняя часть СУБД (ядро) вообще не работает с именами таблиц и их столбцов.

Язык SQL содержит специальные средства определения ограничений целостности БД. Опять же, ограничения целостности хранятся в специальных таблицах-каталогах, и обеспечение контроля целостности БД производится на языковом уровне, т.е. при компиляции операторов модификации БД компилятор SQL на основании имеющихся в БД ограничений целостности генерирует соответствующий программный код.

Специальные операторы языка SQL позволяют определять так называемые представления БД, фактически являющиеся хранимыми в БД запросами (результатом любого запроса к реляционной БД является таблица) с именованными столбцами. Для пользователя представление является такой же таблицей, как любая базовая таблица, хранимая в БД, но с помощью представлений можно ограничить или наоборот расширить видимость БД для конкретного пользователя. Поддержание представлений производится также на языковом уровне.

Наконец, авторизация доступа к объектам БД производится также на основе специального набора операторов SQL. Идея состоит в том, что для выполнения операторов SQL разного вида пользователь должен обладать различными полномочиями. Пользователь, создавший таблицу БД, обладает полным набором полномочий для работы с этой таблицей. В число этих полномочий входит полномочие на передачу всех или части полномочий другим пользователям, включая полномочие на передачу полномочий. Полномочия пользователей описываются в специальных таблицах-каталогах, контроль полномочий поддерживается на языковом уровне.

Модель данных, или **концептуальное описание предметной области** - самый абстрактный уровень проектирования баз данных.

Структуры данных

База данных, организованная с помощью инвертированных списков, похожа на реляционную БД, но с тем отличием, что хранимые таблицы и пути доступа к ним видны пользователям. При этом:

a. Строки таблиц упорядочены системой в некоторой физической последовательности.
b. Физическая упорядоченность строк всех таблиц может определяться и для всей БД (так делается, например, в Datacom/DB).
c. Для каждой таблицы можно определить произвольное число ключей поиска, для которых строятся индексы. Эти индексы автоматически поддерживаются системой, но явно видны пользователям.

Иерархические базы данных

В основе данной модели - иерархическая модель данных. В этой модели имеется один главный объект и остальные - подчиненные - объекты, находящиеся на разных уровнях иерархии. Взаимосвязи объектов образуют иерархическое дерево с одним корневым объектом.

Иерархическая БД состоит из упорядоченного набора нескольких экземпляров одного типа дерева. Автоматически поддерживается целостность ссылок между предками и потомками. Основное правило: никакой потомок не может существовать без своего родителя (рис. 1).

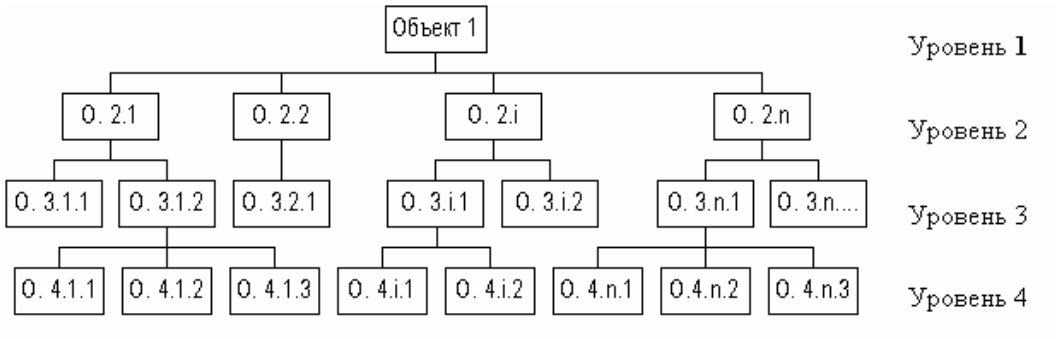

Рис. 1. Схема иерархической модели данных

Типичным представителем (наиболее известным и распространенным) является Information Management System (IMS) фирмы IBM. Первая версия появилась в 1968 г. До сих пор поддерживается много баз данных этой системы.

Сетевые базы данных

Сетевой подход к организации данных является расширением иерархического. В иерархических структурах запись-потомок должна иметь в точности одного предка; в сетевой структуре данных потомок может иметь любое число предков.

В сетевой модели данных любой объект может быть одновременно и главным, и подчиненным, и может участвовать в образовании любого числа взаимосвязей с другими объектами. Сетевая БД состоит из набора записей и набора связей между этими записями, а если говорить более точно - из набора экземпляров каждого типа из заданного в схеме БД набора типов записи и набора экземпляров каждого типа из заданного набора типов связи (рис. 2.).

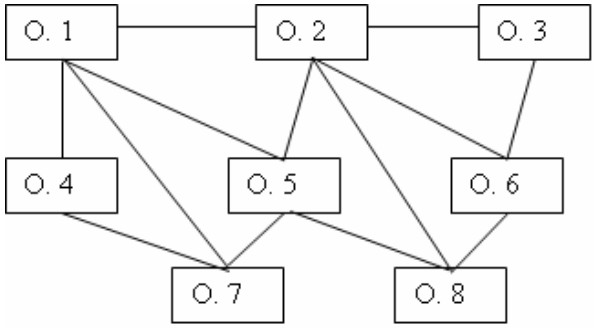

Рис. 2. Схема сетевой модели

Типичным представителем является **Integrated Database Management System (IDMS)** компании Cullinet Software, Inc., предназначенная для использования на машинах основного класса фирмы IBM под управлением большинства операционных систем. Архитектура системы основана на предложениях Data Base Task Group (DBTG) Комитета по языкам программирования Conference on Data Systems Languages (CODASYL) - организации, ответственной за определение языка программирования Кобол. Отчет DBTG был опубликован в 1971 г., а позже появилось несколько систем, среди которых IDMS.

Тема 2. РЕЛЯЦИОННАЯ МОДЕЛЬ И ЕЕ ХАРАКТЕРИСТИКИ. ЦЕЛОСТНОСТЬ В РЕЛЯЦИОННОЙ МОДЕЛИ

Реляционные системы далеко не сразу получили широкое распространение. В то время как основные теоретические результаты в этой области были получены еще в 70-х годах и тогда же появились первые прототипы реляционных СУБД, долгое время считалось невозможным добиться эффективной реализации таких систем. Однако постепенное накопление методов и алгоритмов организации реляционных баз данных и управления ими привели к тому, что уже в середине 80-х годов реляционные системы практически вытеснили с мирового рынка ранние СУБД.

Реляционная модель данных основывается на математических принципах, вытекающих непосредственно из теории множеств и логики предикатов. Эти принципы впервые были применены в области моделирования данных в конце 1960-х гг. доктором Е.Ф. Коддом, в то время работавшим в IBM, а впервые опубликованы - в 1970 г. [1].

Техническая статья "Реляционная модель данных для больших разделяемых банков данных" доктора Е.Ф. Кодда, опубликованная в 1970 г., является родоначальницей современной теории реляционных БД. Доктор Кодд определил 13 правил реляционной модели (которые называют 12 правилами Кодда).

Атрибуты сущности бывают:

1. **Идентифицирующие и описательные**. Идентифицирующие атрибуты имеют уникальное значение для сущностей данного типа и являются потенциальными ключами. Они позволяют однозначно распознавать экземпляры сущности. Из потенциальных ключей выбирается один **первичный ключ** (ПК). В качестве ПК обычно выбирается потенциальный ключ, по которому чаще происходит обращение к экземплярам записи. ПК должен включать в свой состав минимально необходимое для идентификации количество атрибутов. Остальные атрибуты называются описательными.
2. **Простые и составные.** Простой атрибут состоит из одного компонента, его значение неделимо. Составной атрибут является комбинацией нескольких компонентов, возможно, принадлежащих разным типам данных (например, адрес). Решение о том, использовать составной атрибут или разбивать его на компоненты, зависит от особенностей процессов его использования и может быть связано с обеспечением высокой скорости работы с большими базами данных.

3. **Однозначные и многозначные** - могут иметь соответственно одно или много значений для каждого экземпляра сущности.
4. **Основные и производные**. Значение основного атрибута не зависит от других атрибутов. Значение производного атрибута вычисляется на основе значений других атрибутов (например, возраст человека вычисляется на основе даты его рождения и текущей даты

Спецификация атрибута состоит из его названия, указания типа данных и описания ограничений целостности - множества значений (или домена), которые может принимать данный атрибут.

Домен - это набор всех допустимых значений, которые может содержать атрибут. Понятие "домен" часто путают с понятием "тип данных". Необходимо различать эти два понятия. Тип данных - это физическая концепция, а домен - логическая. Например, "целое число" - это тип данных, а "возраст" - это домен.

Связи - на концептуальном уровне представляют собой простые ассоциации между сущностями. Например, утверждение "Покупатели приобретают продукты" указывает, что между сущностями "Покупатели" и "Продукты" существует связь, и такие сущности называются **участниками** этой связи.

Существует несколько типов связей между двумя сущностями: это связи "**один к одному**", "**один ко многим**" и "**многие ко многим**".

Каждая связь в реляционной модели характеризуется именем, обязательностью, типом и степенью. Различают **факультативные** и **обязательные** связи. Если сущность одного типа оказывается по необходимости связанной с сущностью другого типа, то между этими типами объектов существует обязательная связь (обозначается двойной линией). Иначе связь является факультативной.

Степень связи определяется количеством сущностей, которые охвачены данной связью. Пример бинарной связи - связь между отделом и сотрудниками, которые в нем работают.

Схемой реляционной базы данных называется набор заголовков отношений, входящих в базу данных.

Хотя любое отношение можно изобразить в виде таблицы, нужно четко понимать, что *отношения не являются таблицами*. Это близкие, но не совпадающие понятия.

Тема 3. РЕЛЯЦИОННАЯ АЛГЕБРА

Основная идея реляционной алгебры состоит в том, что коль скоро отношения являются множествами, то средства манипулирования отношениями могут базироваться на традиционных теоретико-множественных операциях, дополненных некоторыми специальными операциями, специфичными для баз данных.

Существует много подходов к определению реляционной алгебры, которые различаются набором операций и способами их интерпретации, но в принципе, более или менее равносильны. Мы опишем немного расширенный начальный вариант алгебры, который был предложен Коддом. В этом варианте набор основных алгебраических операций состоит из восьми операций, которые делятся на два класса - теоретико-множественные операции и специальные реляционные операции. В состав теоретико-множественных операций входят операции:

- объединения отношений;
- пересечения отношений;
- взятия разности отношений;
- прямого произведения отношений.

Специальные реляционные операции включают:

- ограничение отношения;
- проекцию отношения;
- соединение отношений;
- деление отношений.

Кроме того, в состав алгебры включается операция присваивания, позволяющая сохранить в базе данных результаты вычисления алгебраических выражений, и операция переименования атрибутов, дающая возможность корректно сформировать заголовок (схему) результирующего отношения. Если не вдаваться в некоторые тонкости, которые мы рассмотрим в следующих подразделах, то почти все операции предложенного выше набора обладают очевидной и простой интерпретацией.

Поскольку результатом любой реляционной операции (кроме операции присваивания) является некоторое отношение, можно образовывать реляционные выражения, в которых вместо отношения-операнда некоторой реляционной операции находится вложенное реляционное выражение.

Каждое отношение характеризуется схемой (или заголовком) и набором кортежей (или телом). Поэтому, если действительно желать иметь алгебру, операции которой замкнуты относительно понятия отношения, то каждая операция должна производить отношение в полном смысле, т.е. оно должно обладать и телом, и заголовком. Только в этом случае будет действительно возможно строить вложенные выражения.

Заголовок отношения представляет собой множество пар <имя-атрибута, имя-домена>. Если посмотреть на общий обзор реляционных операций, приведенный в предыдущем подразделе, то видно, что домены атрибутов

результирующего отношения однозначно определяются доменами отношений-операндов. Однако с именами атрибутов результата не всегда все так просто.

Аналогичные проблемы могут возникать и в случаях других двуместных операций. Для их разрешения в состав операций реляционной алгебры вводится операция переименования. Ее следует применять в любом случае, когда возникает конфликт именования атрибутов в отношениях - операндах одной реляционной операции. Тогда к одному из операндов сначала применяется операция переименования, а затем основная операция выполняется уже безо всяких проблем.

Начнем с операции объединения (все, что будет говориться по поводу объединения, переносится на операции пересечения и взятия разности). Смысл операции объединения в реляционной алгебре в целом остается теоретико-множественным. Но если в теории множеств операция объединения осмысленна для любых двух множеств-операндов, то в случае реляционной алгебры результатом операции объединения должно являться отношение. Если допустить в реляционной алгебре возможность теоретико-множественного объединения произвольных двух отношений (с разными схемами), то, конечно, результатом операции будет множество, но множество разнотипных кортежей, т.е. не отношение. Если исходить из требования замкнутости реляционной алгебры относительно понятия отношения, то такая операция объединения является бессмысленной.

Другие проблемы связаны с операцией взятия прямого произведения двух отношений. В теории множеств прямое произведение может быть получено для любых двух множеств, и элементами результирующего множества являются пары, составленные из элементов первого и второго множеств. Поскольку отношения являются множествами, то и для любых двух отношений возможно

получение прямого произведения. Но результат не будет отношением! Элементами результата будут являться не кортежи, а пары кортежей.

Поэтому в реляционной алгебре используется специализированная форма операции взятия прямого произведения - расширенное прямое произведение отношений. При взятии расширенного прямого произведения двух отношений элементом результирующего отношения является кортеж, являющийся конкатенацией (или слиянием) одного кортежа первого отношения и одного кортежа второго отношения.

Но теперь возникает второй вопрос - как получить корректно сформированный заголовок отношения-результата? Очевидно, что проблемой может быть именование атрибутов результирующего отношения, если отношения-операнды обладают одноименными атрибутами.

Эти соображения приводят к появлению понятия *совместимости по взятию расширенного прямого произведения*. Два отношения совместимы по взятию прямого произведения в том и только в том случае, если множества имен атрибутов этих отношений не пересекаются. Любые два отношения могут быть сделаны совместимыми по взятию прямого произведения путем применения операции переименования к одному из этих отношений.

Следует заметить, что операция взятия прямого произведения не является слишком осмысленной на практике. Во-первых, мощность ее результата очень велика даже при допустимых мощностях операндов, а во-вторых, результат операции не более информативен, чем взятые в совокупности операнды. Как мы увидим немного ниже, основной смысл включения операции расширенного прямого произведения в состав реляционной алгебры состоит в том, что на ее основе определяется действительно полезная операция соединения.

Тема 4. ПРОЕКТИРОВАНИЕ БД. НОРМАЛЬНЫЕ ФОРМЫ ОТНОШЕНИЙ

Архитектура системы баз данных ANSI/SPARC состоит из 3 уровней

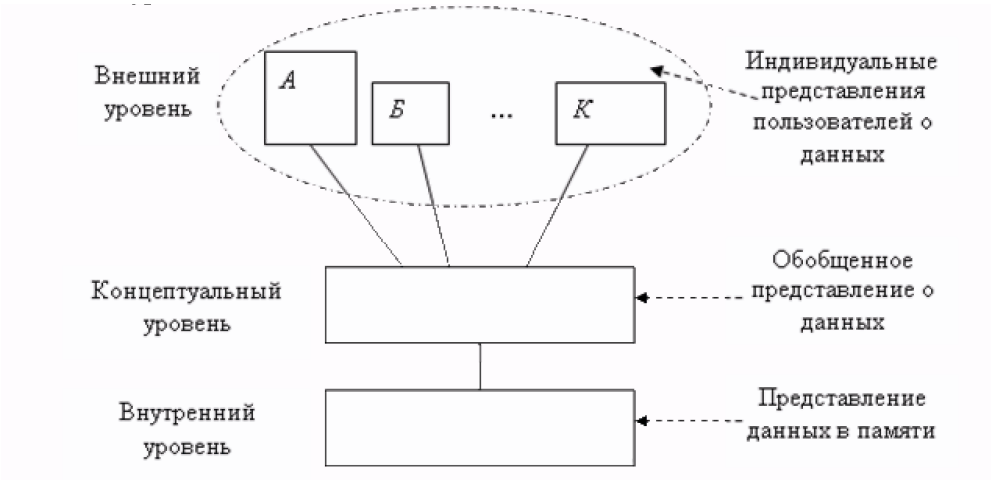

Рис. 1. Принципиальная архитектура системы баз данных

Как видим, в этой архитектуре учитывается, что с базой данных может работать несколько пользователей (в нашем случае – А, Б, … К, …) с разными информационными потребностями.

При этом исходят из того, что любой Банк Данных должен поддерживать разнообразные представления пользователей о Предметной Области.

Предметная область – часть реального мира, представляющая интерес для данного исследования (использования).

Рассмотрим каждый уровень архитектуры подробнее.

Внешний уровень

Отдельного пользователя интересует, как правило, только некоторая часть всей базы данных. Представление отдельного пользователя о предметной области называется **внешним представлением**.

Таким образом, внешний уровень состоит из внешних представлений (которые в английской терминологии называются views) ≈

Внешнее представление – это содержимое базы данных, каким его видит определенный пользователь.

Состоит из множества типов **внешних записей**.

Под **записью** понимается группа взаимосвязанных элементов данных, рассматриваемых как единое целое.

Пример 1.

Пользователь из отдела кадров может рассматривать базу данных как набор записей с информацией об отделах и набор записей с информацией о служащих, и может ничего не знать о записях с информацией о деталях и поставщиках, с которыми работают пользователи из отдела поставок и сбыта.

Для использования компьютера при обработке информации о предметной области эту информацию нужно представлять в специальном виде, строго, формализовано. Формализация - неотъемлемая часть разработки любой программной системы.

Способ формального описания баз данных заключается в использовании **схем**.

Схема - описание структуры БД в формализованном виде.

Схемы используются для строгого, формального описания каждого уровня архитектуры.

На внешнем уровне каждое представление пользователя описывается с помощью **внешней схемы**. Для внешних схем в общем случае используется собственный язык.

Концептуальный уровень

Концептуальное представление формируется на основе интеграции внешних представлений пользователей.

Концептуальное представление—представление **всего** содержимого БД.

Как правило, концептуальное представление существенно отличается от внешних представлений отдельных пользователей (поскольку суммирует их разрозненные представления в одно обобщенное), и состоит из множества типов **концептуальных записей**.

Концептуальное представление определяется с помощью **концептуальной схемы**.

Концептуальная схема – описание полной общей логической структуры базы данных

Концептуальная схема использует (в общем случае) другой язык описания данных. Определения концептуального языка должны относиться **только** к содержанию данных, не касаясь физических подробностей их хранения.

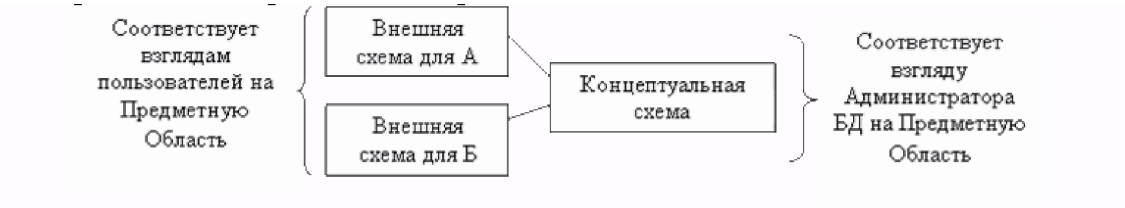

Рис. 2. Соответствие между взглядами пользователей и взглядом администратора БД

В концептуальной схеме **не рассматриваются** способы организации хранения или методы доступа к хранимым данным.

Определения в концептуальной схеме, помимо описания типов записей, могут включать такие средства, как безопасность, правила поддержания целостности.

Записи концептуального уровня не обязаны совпадать с записями внешних уровней.

Внутренний уровень

Внутреннее представление БД — представление структуры хранения записей, состоит из множества типов **хранимых записей**.

Внутреннее представление так же не связано с физическим уровнем, т.е. не рассматриваются физические записи, физические устройства хранения (например, цилиндры и дорожки цилиндры и дорожки), способы доступа к данным, расположенным удаленно.

Внутреннее представление описывается с помощью **внутренней схемы**, которая определяет не только различные типы хранимых записей, но и существующие индексы, способы представления хранимых полей, и т.д.

Внутренняя схема использует внутренний язык определения данных. Записи внутреннего уровня чаще всего не совпадают с записями внешних и концептуального уровней.

Детализованная архитектура системы БД

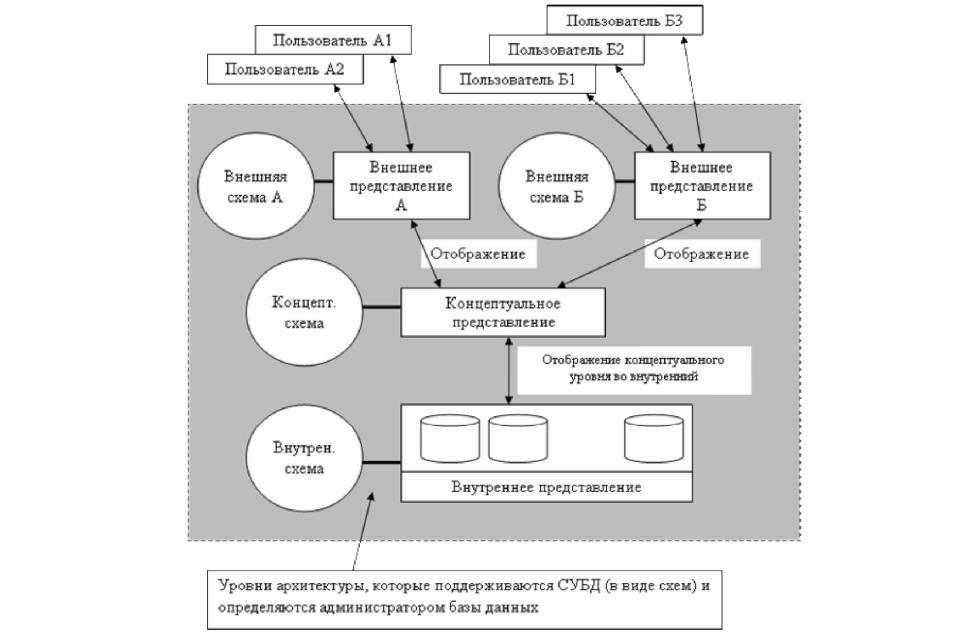

Рис. 3. Детализованная архитектура системы баз данных

Как видно, выбор СУБД определяет способ представления внешних, концептуальной и внутренней схем. Ответственность за корректное представление этих схем, а также за управление данными и схемами несет специальная категория пользователей – администраторы баз данных.

Группа администратора базы данных (АБД)

Поскольку система баз данных может быть весьма большой и может иметь много пользователей, должно существовать лицо или группа лиц, управляющих этой системой. Такое лицо называется **администратором базы данных** (АБД).

В любой базе данных должен быть хотя бы один человек, выполняющий административные обязанности; если база данных большая, эти обязанности могут быть распределены между несколькими администраторами.

Обязанности администратора базы данных

В обязанности администратора могут входить:

- инсталляция и обновление версий СУБД и прикладных инструментов
- распределение дисковой памяти и планирование будущих требований системы к памяти
- создание первичных структур памяти в базе данных (табличных пространств) по мере проектирования приложений разработчиками приложений
- создание первичных объектов (таблиц, представлений, индексов) по мере проектирования приложений разработчиками
- модификация структуры базы данных в соответствии с потребностями приложений
- зачисление пользователей и поддержание защиты системы
- соблюдение лицензионного соглашения по СУБД
- управление и отслеживание доступа пользователей к базе данных
- отслеживание и оптимизация производительности базы данных
- планирование резервного копирования и восстановления
- поддержание архивных данных на устройствах хранения информации
- осуществление резервного копирования и восстановления
- обращение за техническим сопровождением к разработчикам СУБД

Обязанности сотрудника службы безопасности

В некоторых случаях база данных должна также иметь одного или нескольких сотрудников службы безопасности. Сотрудник службы безопасности главным образом отвечает за регистрацию новых пользователей, управление и отслеживание доступа пользователей к базе данных, и защиту базы данных.

Обязанности разработчика приложений

В обязанности разработчика приложений входит:

• проектирование и разработка приложений базы данных

• проектирование структуры базы данных в соответствии с требованиями приложений

• оценка требований памяти для приложения

• формулирование модификаций структуры базы данных для приложения

• передача вышеупомянутой информации администратору базы данных

• настройка приложения в процессе его разработки

• установка мер по защите приложения в процессе его разработки

Основные логические объекты базы данных в современной СУБД.

Несмотря на достаточно простую архитектуру СБД в целом, на практике СУБД поддерживает более детальную классификацию объектов СБД, чем внешние, концептуальная и внутренняя схемы.

СУБД считает объектами СБД все доступные пользователю непосредственно компоненты структуры данных. Набор объектов обычно отличается от СУБД к СУБД, и даже от версии к версии одной СУБД. Чаще всего различают два набора объектов. Один набор объектов ведет начало от СУБД ORACLE, второй – предложен Microsoft.

Эти наборы объектов отличаются способом введения значений, за приращением которых следит СУБД – т.н. счетчики, или последовательности.

Рассмотрим набор объектов СБД, который поддерживается MS SQL Server 2000.

Таблица (table) – единственный объект СБД, в котором реально хранятся данные; являются двумерными матрицами.

Хранимая процедура (stored procedure) – набор команд SQL и Transact-SQL, сохраненный под определенным именем. Пользователи работают с таким набором, как с единым целым, обращаясь к нему по имени.

Триггер (trigger) – специальный вид хранимых процедур, который автоматически запускается сервером при удалении, изменении или добавлении записи в таблицу.

Представление (view) – виртуальная таблица, отражает результат запроса на выборку. Пользователь может работать с представлением как с обычной таблицей. С помощью представлений обычно описывается внешняя схема для некоторой группы пользователей.

Индекс (index) – файл специального вида, предназначен для повышения скорости работы с данными (например, для ускорения поиска данных) за счет представления этих данных в упорядоченном виде

Пользовательский тип данных (user-defined datatype) – тип данных, созданный пользователем на основе встроенных типов данных СУБД

Функция пользователя (user-defined function) – функция, создаваемая пользователем. Как и хранимая процедура, функция – тоже набор команд. В отличие от хранимой процедуры, функция может быть использована в выражениях (как, например, функция вычисления квадратного корня из значения поля).

Ограничение целостности (constraint) – объект базы данных, контролирует логическую целостность данных

Умолчание (default) – объект базы данных, который также как и ограничение целостности, может привязываться к столбцу таблицы или к пользовательскому типу данных.

Набор объектов, который поддерживается ORACLE, отличается от набора MS SQL Server в следующем:

Появляется объект «последовательность»:

Последовательность (sequence) - генерирует уникальные порядковые номера, которые могут использоваться как значения числовых столбцов таблиц базы данных.

Последовательности упрощают прикладное программирование, автоматически генерируя уникальные числовые значения для строк одной или нескольких таблиц. Номера, генерируемые последовательностью, независимы от таблиц, так что одну и ту же последовательность можно использовать для нескольких таблиц. После ее создания, к последовательности могут обращаться различные пользователи, чтобы получать действительные порядковые номера.

Под названием «программная единица» объединяют процедуры, функции и пакеты.

Процедура (procedure) – набор команд SQL и PL/SQL (языка программирования для ORACLE), который исполняется как единое целое и не возвращает результатов.

Функция (function) – набор команд SQL и PL/SQL (языка программирования для ORACLE), который исполняется как единое целое и возвращает результаты.

Пакет (package) – набор процедур и функций, сохраненный под некоторым именем.

Синоним (synonym) – это алиас (дополнительное имя) для таблицы, представлений, последовательности или программной единицы. Синоним не есть объект, но он является прямой ссылкой на объект.

Синонимы используются для:

1. маскировки действительного имени и владельца объекта
2. обеспечения общего доступа к объекту
3. достижения прозрачности местоположения для таблиц, представлений или программных единиц удаленной базы данных
4. упрощения кодирования предложений SQL для пользователей базы данных

Синоним может быть общим или личным. Индивидуальный пользователь может создать **личный синоним**, который доступен только этому пользователю. Администраторы баз данных чаще всего создают **общие синонимы**, благодаря которым объекты базовых схем становятся доступными для общего пользования всем пользователям базы данных.

Кластер (cluster) – это группа из одной или нескольких таблиц, физически хранящихся вместе. В кластеризованных таблицах связанные данные хранятся вместе, более эффективно. В некластеризованных таблицах связанные данные хранятся отдельно, занимая больше места.

Связь баз данных (database link) – именованный объект, который описывает "путь" от одной базы данных к другой. Связи баз данных неявно используются при обращении к **глобальному имени объекта** в распределенной базе данных.

Для того, чтобы управлять отдельными объектами базы данных, СУБД группирует всю информацию об этих объектах в **схеме** (schema). Схема содержит детальное описание свойств всех созданных в БД объектов.

Основные физические объекты базы данных в современной СУБД.

Все объекты и данные одной базы данных хранятся в одном или нескольких файлах данных.

При планировании базы данных планируется и размещение данных по файлам. Каждая база данных состоит минимум из двух файлов – **файла данных** и **файла для хранения журнала транзакций** (действий с базой данных).

Каждый **файл данных** рассматривается СУБД как **набор страниц**. **Страница** – это блок фиксированного размера, который считывается с диска за одну операцию чтения. Размер страницы составляет несколько килобайт, например, в MS SQL Server 2000 размер страницы составляет 8 Кб.

Различают страницы типов **data** (только для хранения данных), **index** (для данных индекса), **text/image** (для хранения данных типа текста, изображения), **Global Allocation Map** (GAM, для хранения информации об использовании

групп страниц – какому файлу какая группа страниц принадлежит), **Page Free Space** (для свободных страниц), **Index Allocation Map** (IAM, для хранения информации о самих группах страниц).

Для более эффективного управления страницами СУБД обычно объединяет группу страниц в **экстент** (extent). Размер экстента, например, в MS SQL Server 2000 – 8 страниц.

При проектировании базы данных решаются две основных проблемы:

- Каким образом отобразить объекты предметной области в абстрактные объекты модели данных, чтобы это отображение не противоречило семантике предметной области и было по возможности лучшим (эффективным, удобным и т.д.)? Часто эту проблему называют проблемой логического проектирования баз данных.
- Как обеспечить эффективность выполнения запросов к базе данных, т.е. каким образом, имея в виду особенности конкретной СУБД, расположить данные во внешней памяти, создание каких дополнительных структур (например, индексов) потребовать и т.д.? Эту проблему называют проблемой физического проектирования баз данных.

В случае реляционных баз данных трудно представить какие-либо общие рецепты по части физического проектирования. Здесь слишком много зависит от используемой СУБД. Например, при работе с СУБД Ingres можно выбирать один из предлагаемых способов физической организации отношений, при работе с System R следовало бы прежде всего подумать о кластеризации отношений и требуемом наборе индексов и т.д. Поэтому мы ограничимся вопросами логического проектирования реляционных баз данных, которые существенны при использовании любой реляционной СУБД.

Более того, мы не будем касаться очень важного аспекта проектирования - определения ограничений целостности (за исключением ограничения первичного ключа). Дело в том, что при использовании СУБД с развитыми механизмами ограничений целостности (например, SQL-ориентированных

систем) трудно предложить какой-либо общий подход к определению ограничений целостности. Эти ограничения могут иметь очень общий вид, и их формулировка пока относится скорее к области искусства, чем инженерного мастерства. Самое большее, что предлагается по этому поводу в литературе, это автоматическая проверка непротиворечивости набора ограничений целостности.

Так что будем считать, что проблема проектирования реляционной базы данных состоит в обоснованном принятии решений о том,

- из каких отношений должна состоять БД и
- какие атрибуты должны быть у этих отношений.

Понимание принципов разработки, организации и функционирования подобных систем, способов хранения и обработки информации необходимо каждому современному специалисту.

При описании информационной системы предполагается, что она содержит два типа сущностей: операционные сущности, которые выполняют какую-либо обработку (некоторый аналог программы), и пассивные сущности, которые хранят информацию, доступную для пополнения, изменения, поиска, чтения (база данных).

При проектировании сложных информационных систем используется метод декомпозиции - система разбивается на составные части, которые связаны, взаимодействуют друг с другом и образуют иерархическую структуру. Иерархический характер сложных систем хорошо согласуется с принципом групповой разработки. В этом случае деятельность каждого участника проекта ограничивается соответствующим иерархическим уровнем.

Классический подход к разработке сложных систем представляет собой структурное проектирование, при котором осуществляется алгоритмическая декомпозиция системы по методу "**сверху вниз**". Именно в этом случае можно построить хорошо функционирующую систему с общей базой данных,

согласованными форматами использования и обработки информации на всех участках, с оптимальным взаимодействием всех подсистем.

Исторически сложилось так, что некоторые системы разрабатывались по методу "**снизу вверх**": вначале создавались отдельные автоматизированные рабочие места (АРМы), затем предпринимались попытки объединения их в единую информационную систему. Подобные разработки для крупных систем не могут быть успешны.

При создании проекта информационной системы для проектирования ее базы данных следует определить:

1. объекты информационной системы (сущности в концептуальной модели);
2. их свойства (атрибуты);
3. взаимодействие объектов (связи) и информационные потоки внутри и между ними.

При этом очень важен анализ существующей практики реализации информационных процессов и нормативной информации (законов, постановлений правительства, отраслевых стандартов), определяющих необходимый объем и формат хранения и передачи информации. Если радикальной перестройки сложившегося информационного процесса не предвидится, следует учитывать имеющиеся формы хранения и обработки информации в виде журналов, ведомостей, таблиц и т.п. бумажных носителей.

Однако предварительно необходимо выполнить анализ возможности перехода на новые системы учета, хранения и обработки информации, возможно, исходя из имеющихся на рынке программных продуктов-аналогов, разработанных крупными информационными компаниями и частично или полностью соответствующими поставленной задаче.

Схема формирования информационной модели представлена на рис.1.

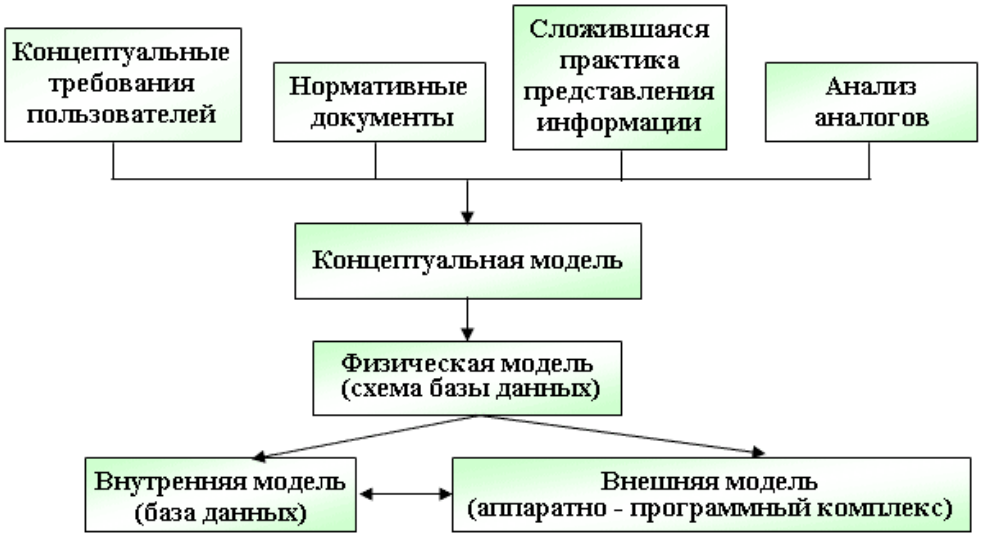

Рис. 1. Схема формирования информационной модели

Концептуальная модель - отображает информационные объекты, их свойства и связи между ними без указания способов физического хранения информации (модель предметной области, иногда ее также называют информационно-логической или инфологической моделью). Информационными объектами обычно являются **сущности** - обособленные объекты или события, информацию о которых необходимо сохранять, имеющие определенные наборы свойств - **атрибутов**.

Физическая модель - отражает все свойства (атрибуты) информационных объектов базы и связи между ними с учетом способа их хранения - используемой СУБД.

Проектирование реляционных баз данных с использованием нормализации

Сначала будет рассмотрен классический подход, при котором весь процесс проектирования производится в терминах реляционной модели данных методом последовательных приближений к удовлетворительному набору схем отношений. Исходной точкой является представление предметной области в виде одного или нескольких отношений, и на каждом шаге проектирования производится некоторый набор схем отношений, обладающих лучшими

свойствами. Процесс проектирования представляет собой процесс нормализации схем отношений, причем каждая следующая нормальная форма обладает свойствами лучшими, чем предыдущая.

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений. Примером набора ограничений является ограничение первой нормальной формы - значения всех атрибутов отношения атомарны. Поскольку требование первой нормальной формы является базовым требованием классической реляционной модели данных, мы будем считать, что исходный набор отношений уже соответствует этому требованию.

В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

Основные свойства нормальных форм:

- каждая следующая нормальная форма в некотором смысле лучше предыдущей;
- при переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются.

В основе процесса проектирования лежит метод нормализации, декомпозиция отношения, находящегося в предыдущей нормальной форме, в

два или более отношения, удовлетворяющих требованиям следующей нормальной формы.

Наиболее важные на практике нормальные формы отношений основываются на фундаментальном в теории реляционных баз данных понятии *функциональной зависимости*. Для дальнейшего изложения нам потребуются несколько определений.

Функциональная зависимость

В отношении R атрибут Y функционально зависит от атрибута X (X и Y могут быть составными) в том и только в том случае, если каждому значению X соответствует в точности одно значение Y: R.X (r) R.Y.

Полная функциональная зависимость

Функциональная зависимость R.X (r) R.Y называется полной, если атрибут Y не зависит функционально от любого точного подмножества X.

Транзитивная функциональная зависимость

Функциональная зависимость R.X −> R.Y называется транзитивной, если существует такой атрибут Z, что имеются функциональные зависимости R.X −> R.Z и R.Z −> R.Y и отсутствует функциональная зависимость R.Z --> R.X. (При отсутствии последнего требования мы имели бы "неинтересные" транзитивные зависимости в любом отношении, обладающем несколькими ключами.)

Неключевой атрибут

Неключевым атрибутом называется любой атрибут отношения, не входящий в состав первичного ключа (в частности, первичного).

Взаимно независимые атрибуты

Два или более атрибута взаимно независимы, если ни один из этих атрибутов не является функционально зависимым от других.

Вторая нормальная форма

Вторая нормальная форма (в этом определении предполагается, что единственным ключом отношения является первичный ключ)

Отношение R находится во второй нормальной форме (2NF) в том и только в том случае, когда находится в 1NF, и каждый неключевой атрибут полностью зависит от первичного ключа.

Если допустить наличие нескольких ключей, то определение 6 примет следующий вид:

Отношение R находится во второй нормальной форме (2NF) в том и только в том случае, когда оно находится в 1NF, и каждый неключевой атрибут полностью зависит от каждого ключа R.

Третья нормальная форма

Третья нормальная форма. (Снова определение дается в предположении существования единственного ключа.)

Отношение R находится в третьей нормальной форме (3NF) в том и только в том случае, если находится в 2NF и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

Отношение R находится в третьей нормальной форме (3NF) в том и только в том случае, если находится в 1NF, и каждый неключевой атрибут не является транзитивно зависимым от какого-либо ключа R.

На практике третья нормальная форма схем отношений достаточна в большинстве случаев, и приведением к третьей нормальной форме процесс проектирования реляционной базы данных обычно заканчивается. Однако иногда полезно продолжить процесс нормализации.

Нормальная форма Бойса-Кодда

Детерминант

Детерминант - любой атрибут, от которого полностью функционально зависит некоторый другой атрибут.

Нормальная форма Бойса-Кодда

Отношение R находится в нормальной форме Бойса-Кодда (BCNF) в том и только в том случае, если каждый детерминант является возможным ключом.

Четвертая нормальная форма

Многозначные зависимости

В отношении R (A, B, C) существует многозначная зависимость R.A −> −> R.B в том и только в том случае, если множество значений B, соответствующее паре значений A и C, зависит только от A и не зависит от C.

Теорема Фейджина

Отношение R (A, B, C) можно спроецировать без потерь в отношения R1 (A, B) и R2 (A, C) в том и только в том случае, когда существует MVD A −> −> B | C.

Под проецированием без потерь понимается такой способ декомпозиции отношения, при котором исходное отношение полностью и без избыточности восстанавливается путем естественного соединения полученных отношений.

Четвертая нормальная форма

Отношение R находится в четвертой нормальной форме (4NF) в том и только в том случае, если в случае существования многозначной зависимости A −> −> B все остальные атрибуты R функционально зависят от A.

Пятая нормальная форма

Зависимость соединения

Отношение R (X, Y, ..., Z) удовлетворяет зависимости соединения * (X, Y, ..., Z) в том и только в том случае, когда R восстанавливается без потерь путем соединения своих проекций на X, Y, ..., Z.

Пятая нормальная форма

Отношение R находится в пятой нормальной форме (нормальной форме проекции-соединения - PJ/NF) в том и только в том случае, когда любая зависимость соединения в R следует из существования некоторого возможного ключа в R.

Пятая нормальная форма - это последняя нормальная форма, которую можно получить путем декомпозиции. Ее условия достаточно нетривиальны, и на практике 5NF не используется. Заметим, что зависимость соединения является обобщением как многозначной зависимости, так и функциональной зависимости.

Тема 6. ПРОЕКТИРОВАНИЕ БД МЕТОДОМ СУЩНОСТЬ-СВЯЗЬ. ER-ДИАГРАММЫ

Модель "Сущность-Связи" (часто ее называют кратко ER-моделью).

На использовании разновидностей ER-модели основано большинство современных подходов к проектированию баз данных (главным образом, реляционных). Модель была предложена Ченом (Chen) в 1976 г. Моделирование предметной области базируется на использовании графических диаграмм, включающих небольшое число разнородных компонентов. В связи с наглядностью представления концептуальных схем баз данных ER-модели получили широкое распространение в системах CASE, поддерживающих автоматизированное проектирование реляционных баз данных. Среди множества разновидностей ER-моделей одна из наиболее развитых

применяется в системе CASE фирмы ORACLE. Ее мы и рассмотрим. Более точно, мы сосредоточимся на структурной части этой модели.

Основными понятиями ER-модели являются сущность, связь и атрибут.

Сущность - это реальный или представляемый объект, информация о котором должна сохраняться и быть доступна. В диаграммах ER-модели сущность представляется в виде прямоугольника, содержащего имя сущности. При этом имя сущности - это имя типа, а не некоторого конкретного экземпляра этого типа. Для большей выразительности и лучшего понимания имя сущности может сопровождаться примерами конкретных объектов этого типа.

Каждый экземпляр сущности должен быть отличим от любого другого экземпляра той же сущности (это требование в некотором роде аналогично требованию отсутствия кортежей-дубликатов в реляционных таблицах).

Связь - это графически изображаемая ассоциация, устанавливаемая между двумя сущностями. Эта ассоциация всегда является бинарной и может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь). В любой связи выделяются два конца (в соответствии с существующей парой связываемых сущностей), на каждом из которых указывается имя конца связи, степень конца связи (сколько экземпляров данной сущности связывается), обязательность связи (т.е. любой ли экземпляр данной сущности должен участвовать в данной связи).

Связь представляется в виде линии, связывающей две сущности или ведущей от сущности к ней же самой. При это в месте "стыковки" связи с сущностью используются трехточечный вход в прямоугольник сущности, если для этой сущности в связи могут использоваться много (many) экземпляров сущности, и одноточечный вход, если в связи может участвовать только один экземпляр сущности. Обязательный конец связи изображается сплошной линией, а необязательный - прерывистой линией.

Как и сущность, связь - это типовое понятие, все экземпляры обеих пар связываемых сущностей подчиняются правилам связывания.

Существует несколько типов связей между двумя сущностями: это связи "**один к одному**", "**один ко многим**" и "**многие ко многим**".

Каждая связь в реляционной модели характеризуется именем, обязательностью, типом и степенью. Различают **факультативные** и **обязательные** связи. Если сущность одного типа оказывается по необходимости связанной с сущностью другого типа, то между этими типами объектов существует обязательная связь (обозначается двойной линией). Иначе связь является факультативной.

Степень связи определяется количеством сущностей, которые охвачены данной связью. Пример бинарной связи - связь между отделом и сотрудниками, которые в нем работают.

Важнейшая цель проектирования информационной модели - выработка непротиворечивой структурированной интерпретации реально существующей информации изучаемой предметной области и взаимодействия между ее структурными компонентами

Понятие концептуальной модели данных связано с методологией семантического моделирования данных, т.е. с представлением данных в контексте их взаимосвязей с другими данными.

Методология IDEF1X - один из подходов к семантическому моделированию данных, основанный на концепции "сущность-связь" (Entity-Relationship). Это инструмент для анализа информационной структуры систем различной природы. Информационная модель, построенная с помощью IDEF1X-методологии, отображает логическую структуру информации об объектах системы

Таким образом, концептуальная модель, представленная в соответствии со стандартом IDEF1X, является логической схемой базы данных для проектируемой системы

Основными объектами концептуальной модели являются сущности и связи.

Сущность - некоторый обособленный объект или событие моделируемой системы, имеющий определенный набор свойств - атрибутов. Отдельный элемент этого множества называется "экземпляром сущности". Сущность может обладать одним или несколькими атрибутами, которые однозначно идентифицируют каждый образец сущности, и может обладать любым количеством связей с другими сущностями.

Правила для атрибутов сущности:

1. Каждый атрибут должен иметь уникальное имя.
2. Сущность может обладать любым количеством атрибутов.
3. Сущность может обладать любым количеством наследуемых атрибутов, но наследуемый атрибут должен быть частью первичного ключа сущности-родителя.
4. Для каждого экземпляра сущности должно существовать значение каждого его атрибута (правило необращения в нуль - Not Null).
5. Ни один из экземпляров сущности не может обладать более чем одним значением для ее атрибута.

Сущность изображается на ER-диаграмме в виде прямоугольника, в верхней части которого приводится ее название; далее следует список атрибутов. Ключевые атрибуты могут быть выделены подчеркиванием или иным способом.

Стандарт IDEF1X описывает способы изображения двух типов сущностей - независимой и зависимой, и связей - идентифицирующих и

неидентифицирующих Каждая сущность может обладать любым количеством связей с другими сущностями.

Сущность является независимой, если каждый ее экземпляр может быть однозначно идентифицирован без определения его связей с другими сущностями.

Сущность называется зависимой, если однозначная идентификация ее экземпляра зависит от его связей с другими сущностями.

Сущность может обладать атрибутами, которые наследуются через связь с родительской сущностью. Последние обычно являются **внешними ключами** и служат для организации связей между сущностями. Если **внешний ключ** сущности используется в качестве ее первичного ключа (PK) или как часть составного первичного ключа, то сущность является зависимой от родительской сущности. Если внешний ключ не является первичным и не входит в составной первичный ключ, то сущность является независимой от родительской сущности.

Если сущность является зависимой, то связь ее с родительской сущностью называется идентифицирующей, в противном случае - неидентифицирующей.

Связь изображается на ER-диаграмме линией, проводимой между сущностью-родителем и сущностью-потомком с точкой на конце линии у сущности-потомка. идентифицирующая связь изображается сплошной линией, неидентифицирующая - пунктирной.

Связи дается имя, выражаемое грамматической формой глагола. Для связи дополнительно может присутствовать указание мощности: какое количество экземпляров сущности-потомка может существовать для сущности-родителя. Имя связи всегда формируется с точки зрения родителя, так что может быть образовано предложение, если соединить имя сущности родителя,

имя связи, выражение мощности и имя сущности-потомка (например "много СТУДЕНТов - сдают - ЭКЗАМЕН").

Принципы изображения концептуальных моделей баз данных стандарта IDEF1 и IDEF1X используют CASE Studio и другие CASE-средства. Подобные системы позволяют на основе концептуальной модели генерировать физическую модель и программный код создания базы данных для большинства наиболее распространенных СУБД и серверов баз данных.

Семантическое моделирование данных, ER-диаграммы

Широкое распространение реляционных СУБД и их использование в самых разнообразных приложениях показывает, что реляционная модель данных достаточна для моделирования предметных областей. Однако проектирование реляционной базы данных в терминах отношений на основе кратко рассмотренного нами механизма нормализации часто представляет собой очень сложный и неудобный для проектировщика процесс.

При этом проявляется ограниченность реляционной модели данных в следующих аспектах:

- Модель не предоставляет достаточных средств для представления смысла данных. Семантика реальной предметной области должна независимым от модели способом представляться в голове проектировщика. В частности, это относится к упоминавшейся нами проблеме представления ограничений целостности.
- Для многих приложений трудно моделировать предметную область на основе плоских таблиц. В ряде случаев на самой начальной стадии проектирования проектировщику приходится производить насилие над собой, чтобы описать предметную область в виде одной (возможно, даже ненормализованной) таблицы.
- Хотя весь процесс проектирования происходит на основе учета зависимостей, реляционная модель не предоставляет каких-либо средств для представления этих зависимостей.

- Несмотря на то, что процесс проектирования начинается с выделения некоторых существенных для приложения объектов предметной области ("сущностей") и выявления связей между этими сущностями, реляционная модель данных не предлагает какого-либо аппарата для разделения сущностей и связей.

Семантические модели данных

Потребности проектировщиков баз данных в более удобных и мощных средствах моделирования предметной области вызвали к жизни направление семантических моделей данных. При том, что любая развитая семантическая модель данных, как и реляционная модель, включает структурную, манипуляционную и целостную части, главным назначением семантических моделей является обеспечение возможности выражения семантики данных.

Прежде, чем мы коротко рассмотрим особенности одной из распространенных семантических моделей, остановимся на их возможных применениях.

Наиболее часто на практике семантическое моделирование используется на первой стадии проектирования базы данных. При этом в терминах семантической модели производится концептуальная схема базы данных, которая затем вручную преобразуется к реляционной (или какой-либо другой) схеме. Этот процесс выполняется под управлением методик, в которых достаточно четко оговорены все этапы такого преобразования.

Менее часто реализуется автоматизированная компиляция концептуальной схемы в реляционную. При этом известны два подхода: на основе явного представления концептуальной схемы как исходной информации для компилятора и построения интегрированных систем проектирования с автоматизированным созданием концептуальной схемы на основе интервью с экспертами предметной области. И в том, и в другом случае в результате производится реляционная схема базы данных в третьей нормальной форме

(более точно следовало бы сказать, что автору неизвестны системы, обеспечивающие более высокий уровень нормализации).

Наконец, третья возможность, которая еще не вышла (или только выходит) за пределы исследовательских и экспериментальных проектов, - это работа с базой данных в семантической модели, т.е. СУБД, основанные на семантических моделях данных. При этом снова рассматриваются два варианта: обеспечение пользовательского интерфейса на основе семантической модели данных с автоматическим отображением конструкций в реляционную модель данных (это задача примерно такого же уровня сложности, как автоматическая компиляция концептуальной схемы базы данных в реляционную схему) и прямая реализация СУБД, основанная на какой-либо семантической модели данных. Наиболее близко ко второму подходу находятся современные объектно-ориентированные СУБД, модели данных которых по многим параметрам близки к семантическим моделям (хотя в некоторых аспектах они более мощны, а в некоторых - более слабы).

Основные понятия модели Entity-Relationship (Сущность-Связи)

В первой нормальной форме ER-схемы устраняются повторяющиеся атрибуты или группы атрибутов, т.е. производится выявление неявных сущностей, "замаскиро-ванных" под атрибуты.

Во второй нормальной форме устраняются атрибуты, зависящие только от части уникального идентификатора. Эта часть уникального идентификатора определяет отдельную сущность.

В третьей нормальной форме устраняются атрибуты, зависящие от атрибутов, не входящих в уникальный идентификатор. Эти атрибуты являются основой отдельной сущности.

Получение реляционной схемы из ER-схемы

Шаг 1. Каждая простая сущность превращается в таблицу. Простая сущность - сущность, не являющаяся подтипом и не имеющая подтипов. Имя сущности становится именем таблицы.

Шаг 2. Каждый атрибут становится возможным столбцом с тем же именем; может выбираться более точный формат. Столбцы, соответствующие необязательным атрибутам, могут содержать неопределенные значения; столбцы, соответствующие обязательным атрибутам, - не могут.

Шаг 3. Компоненты уникального идентификатора сущности превращаются в первичный ключ таблицы. Если имеется несколько возможных уникальных идентификатора, выбирается наиболее используемый. Если в состав уникального идентификатора входят связи, к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (этот процесс может продолжаться рекурсивно). Для именования этих столбцов используются имена концов связей и/или имена сущностей.

Шаг 4. Связи многие-к-одному (и один-к-одному) становятся внешними ключами. Т.е. делается копия уникального идентификатора с конца связи "один", и соответствующие столбцы составляют внешний ключ. Необязательные связи соответствуют столбцам, допускающим неопределенные значения; обязательные связи - столбцам, не допускающим неопределенные значения.

Шаг 5. Индексы создаются для первичного ключа (уникальный индекс), внешних ключей и тех атрибутов, на которых предполагается в основном базировать запросы.

Шаг 6. Если в концептуальной схеме присутствовали подтипы, то возможны два способа:

- все подтипы в одной таблице (а)

- для каждого подтипа - отдельная таблица (б)

При применении способа (а) таблица создается для наиболее внешнего супертипа, а для подтипов могут создаваться представления. В таблицу добавляется по крайней мере один столбец, содержащий код ТИПА; он становится частью первичного ключа.

При использовании метода (б) для каждого подтипа первого уровня (для более нижних - представления) супертип воссоздается с помощью представления UNION (из всех таблиц подтипов выбираются общие столбцы - столбцы супертипа).

Шаг 7. Имеется два способа работы при наличии исключающих связей:

- общий домен (а)
- явные внешние ключи (б)

Если остающиеся внешние ключи все в одном домене, т.е. имеют общий формат (способ (а)), то создаются два столбца: идентификатор связи и идентификатор сущности. Столбец идентификатора связи используется для различения связей, покрываемых дугой исключения. Столбец идентификатора сущности используется для хранения значений уникального идентификатора сущности на дальнем конце соответствующей связи.

Если результирующие внешние ключи не относятся к одному домену, то для каждой связи, покрываемой дугой исключения, создаются явные столбцы внешних ключей; все эти столбцы могут содержать неопределенные значения.

Тема 7. ЯЗЫК SQL

Язык для взаимодействия с БД SQL появился в середине 70-х и был разработан в рамках проекта экспериментальной реляционной СУБД System R. Исходное название языка SEQUEL (Structered English Query Language) только частично отражает суть этого языка. Конечно, язык был ориентирован главным образом на удобную и понятную пользователям формулировку запросов к

реляционной БД, но на самом деле уже являлся полным языком БД, содержащим помимо операторов формулирования запросов и манипулирования БД средства определения и манипулирования схемой БД; определения ограничений целостности и триггеров; представлений БД; возможности определения структур физического уровня, поддерживающих эффективное выполнение запросов; авторизации доступа к отношениям и их полям; точек сохранения транзакции и откатов. Таким образом, SQL стал достаточно мощным языком для взаимодействия с СУБД. На сегодняшний день SQL является единственным стандартным языком запросов. Язык SQL обладает следующими достоинствами:

1. независимость от конкретных СУБД. Если при создании БД не использовались нестандартные возможности языка SQL предоставляемые некоторой СУБД, то такую БД можно без изменений перенести на СУБД другого производителя. К сожалению большинство БД используют особенности СУБД, на которой работают, что затрудняет их перенос на другую СУБД без изменений;

2. реляционная основа. Реляционная модель имеет солидный теоретический фундамент. Язык SQL основан на реляционной модели и является единственным языком для реляционных БД;

3. SQL обладает высокоуровневой структурой, напоминающей английский язык.

4. SQL позволяет создавать различные представления данных для различных пользователей;

5. SQL является полноценным языком для работы с БД;

6. стандарты языка SQL. Официальный стандарт языка SQL опубликован ANSI и ISO в 1989 году и значительно расширен в 1992 году.

Основные понятия SQL

Операторы

В SQL используется приблизительно тридцать операторов, каждый из которых "просит" СУБД выполнить определенное действие, например,

прочитать данные, создать таблицу или добавить в таблицу новые данные. Все операторы SQL имеют одинаковую структуру, которая показана на рис. 1.

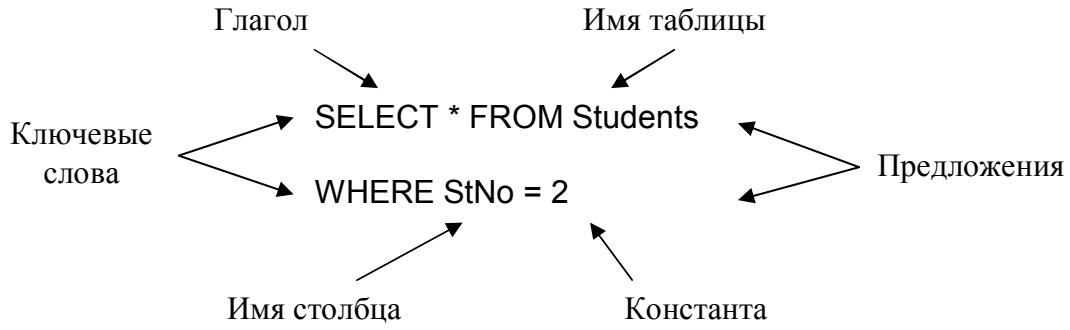

рис. 1 Структура оператора SQL.

Каждый оператор SQL начинается с глагола, т.е. ключевого слова, описывающего действие, выполняемое оператором. Типичными глаголами являются SELECT (выбрать), CREATE (создать), INSERT (добавить), DELETE (удалить), COMMIT(завершить). После глагола идет одно или несколько предложений. Предложение описывает данные, с которыми работает оператор, или содержит уточняющую информацию о действии, выполняемом оператором. Каждое предложение также начинается с ключевого слова, такого как WHERE (где), FROM (откуда), INTO (куда) и HAVING (имеющий). Одни предложения в операторе являются обязательным, а другие – нет. Конкретная структура и содержимое предложения могут изменяться. Многие предложения содержат имена таблиц или столбцов; некоторые из них могут содержать дополнительные ключевые слова, константы и выражения.

В стандарте ANSI/ISO определены ключевые слова, которые применяются в качестве глаголов и в предложениях операторов. В соответствии со стандартом, эти ключевые слова нельзя использовать для именования объектов базы данных, таких как таблицы, столбцы и пользователи

Имена.

У каждого объекта в базе данных есть уникальное имя. Имена используются в операторах SQL и указывают, над каким объектом базы данных оператор должен выполнить действие. В стандарте ANSI/ISO определено, что имена имеются у таблиц, столбцов и пользователей. Во многих реализациях SQL поддерживаются также дополнительные именованные объекты, такие как хранимые процедуры, именованные отношения "первичный ключ – внешний ключ" и формы для ввода данных.

В соответствии со стандартом ANSI/ISO, в SQL имена должны содержать от 1 до 18 символов, начинаться с буквы и не содержать пробелы или специальные символы пунктуации. В стандарте SQL2 максимальное число символов в имени увеличено до 128.

Полное имя таблицы состоит из имени владельца таблицы и собственно ее имени, разделенных точкой (.). Например, полное имя таблицы Students, владельцем которой является пользователь по имени Admin, имеет следующий вид:

Admin.Students

Если в операторе задается имя столбца, SQL сам определяет, в какой из указанных в этом же операторе таблиц содержится данный столбец. Однако если в оператор требуется включить два столбца из различных таблиц, но с одинаковыми именами, необходимо указать полные имена столбцов, которые однозначно определяют их местонахождение. Полное имя столбца состоит из имени таблицы, содержащей столбец, и имени столбца (простого имени), разделенных точкой (.). Например, полное имя столбца StName из таблицы Students имеет следующий вид:

Students.StName

Типы данных в SQL

В стандарте ANSI/ISO определены типы данных, которые можно использовать для представления информации в реляционной базе данных.

Типы данных, имеющиеся в стандарте SQL1, составляют лишь минимальный набор и поддерживаются во всех коммерческих СУБД. В табл. 1 перечислены типы данных, определенные в стандартах SQL1 и SQL2.

табл. 1 Типы данных в SQL.

Тип данных	Описание
CHAR (длина)	Строки символов постоянной длины
CHARACTER (длина)	
VARCHAR(длина)	Строки символов переменной длины*
CHAR VARYING(длина)	
CHARACTER VARYING (длина)	
NCHAP(длина)	Строки локализованных символов постоянной длины*
NATIONAL CHAR(длина)	
NATIONAL CHARACTER(длина)	
NCHAR VARYING(длина)	Строки локализованных символов переменной длины*
NATIONAL CHAR VARYING(длина)	
NATIONAL CHARACTER VARYING(длина)	
INTEGER	Целые числа
INT	
SMALLINT	Маленькие целые числа
BIT(длина)	Строки битов постоянной длины*
BIT VARYNG(длина)	Строки битов переменной длины*
NUMERIC(точность, степень)	Масштабируемые целые (десятичные) числа
DECIMAL(точность, степень)	
DEC(точность, степень)	
FLOAT(точность)	Числа с плавающей запятой
REAL	Числа с плавающей запятой низкой точности
DOUBLE PRECISION	Числа с плавающей запятой высокой точности
DATE	Календарная дата*
TIME(точность)	Время
TIME STAMP(точность)	Дата и время*
INTERVAL	Временной интервал*

В SQL1 используются следующие типы данных:

1. Строки символов постоянной длины. В столбцах, имеющих этот тип данных, обычно хранятся имена людей и компаний, адреса, описания и т д.

2. Целые числа. В столбцах, имеющих этот тип данных, обычно хранятся данные о счетах, количествах, возрастах и т.д. Целочисленные столбцы часто используются также для хранения идентификаторов, таких как идентификатор клиента, служащего или заказа.

3. Масштабируемые целые числа. В столбцах данного типа хранятся числа,) имеющие дробную часть, которые необходимо вычислять точно, например курсы валют и проценты. Кроме того, в таких столбцах часто хранятся) денежные величины.

4. Числа с плавающей запятой. Столбцы этого типа используются для хранения величин, которые можно вычислять приблизительно, например веса и расстояния. Числа с плавающей запятой могут представлять больший диапазон значений, чем десятичные числа, однако при вычислениях могут возникать погрешности округления.

В большинстве коммерческих СУБД помимо типов данных, определенных в стандарте SQL1, имеется множество дополнительных типов данных, большинство из которых вошло в стандарт SQL2. Ниже перечислены наиболее важные из них:

1. Строки символов переменной длины. Почти во всех СУБД поддерживается тип данных VARCHAR, позволяющий хранить строки символов, длина которых изменяется в некотором диапазоне В стандарте SQL1 были определены строки постоянной длины, которые справа дополняются пробелами.

2. Денежные величины. Во многих СУБД поддерживается тип данных MONEY или CURRENCY, который обычно хранится в виде десятичного числа или числа с плавающей запятой. Наличие отдельного типа данных для представления денежных величин позволяет правильно форматировать их при выводе на экран.

3. Дата и время. Поддержка значений даты/времени также широко распространена в различных СУБД, хотя способы ее реализации довольно сильно отличаются друг от друга. Как правило, над значениями этого типа

данных можно выполнять различные операции. В стандарт SQL2 входит определение типов данных DATE, TIME, TIMESTAMP и INTERVAL, включая поддержку часовых поясов и возможность указания точности, представления времени (например, десятые или сотые доли секунды).

4. Булевы данные. Некоторые СУБД явным образом поддерживают логические значения (TRUE или FALSE).

Константы

В стандарте ANSI/ISO определен формат числовых и строковых констант, или литералов, которые представляют конкретные значения данных. Этот формат используется в большинстве реализации SQL.

Числовые константы. Целые и десятичные константы (известные также под названием точных числовых литералов) в операторах SQL представляются в виде обычных. десятичных чисел с необязательным знаком плюс (+) или минус (−) перед ними:

21 -3752000,00 +497500,8778

Константы с плавающей запятой (известные также под названием приблизительных числовых литералов) определяются с помощью символа E и имеют такой же формат, как и в большинстве языков программирования. Ниже приведены примеры констант с плавающей запятой:

1.5E3 -3.14159E1 2.5E-7 0.783926E21

Символ E читается как "умножить на десять в степени", так что первая константа представляет число "1,5 умножить на десять в степени 3", или 1500.

Строковые константы. В соответствии со стандартом ANSI/ISO, строковые константы в SQL должны быть заключены в одинарные кавычки, как показано в следующих примерах:

Jones, John J.' 'New York' 'Western'

Если необходимо включить в строковую константу одинарную кавычку, вместо нее следует написать две одинарные кавычки.

Константы даты и времени. В реляционных СУБД календарные даты, время и интервалы времени представляются в виде строковых констант. Форматы этих констант в различных СУБД отличаются друг от друга. Кроме того, способы записи времени и даты изменяются в зависимости от страны.

Символьные константы. Кроме пользовательских констант, в SQL существуют специальные символьные константы, возвращающие значения, хранимые в самой СУБД.

В стандарт SQL2 вошли наиболее полезные символьные константы из различных реализации SQL, в частности константы `CURRENT_DATE`, `CURRENT_TIME`, `CURRENT_TIMESTAMP`, а также `USER`, `SESSION_USER` и `SYSTEM_USER` (обратите внимание на знак подчеркивания!).

Выражения в SQL используются для выполнения операций над значениями, считанными из базы данных или используемыми для поиска в базе. данных. Например, в следующем запросе вычисляется процентное соотнесение объема и плана продаж для каждого офиса:

```
SELECT CITY, TARGET, SALES, (SALES/TARGET) * 100
FROM OFFICES
```

В соответствии со стандартом ANSI/ISO, в выражениях можно использовать четыре арифметические операции: сложение (`X + Y`), вычитание (`X - Y`), умножение (`X * Y`) и деление (`X / Y`). Для формирования сложных выражений можно использовать скобки.

Отсутствующие данные (значения NULL). Поскольку база данных представляет собой модель реального мира, отдельные элементы данных в ней неминуемо будут отсутствовать или подходить не для всех сущностей. Для указания на такие данные используеВÖся специальная константа – NULL.

Встроенные функции

В стандарт SQL2 вошли наиболее полезные функции из различных реализации SQL. Эти функции перечислены в таблице.

табл. 2 Встроенные функции SQL.

Функция	Возвращается
BIT LENGTH(строка)	количество битов в строке
CAST(значение AS тип данных)	значение, преобразованное тип данных (например, дата преобразованная в строку)
CHAR_LENGTH(строка)	длина строки символов
CONVERT(строка USING функция)	строка, преобразованная в соответствии с указанной функцией
CURRENT_DATE	текущая дата
CURRENT_TIME(точность)	текущее время с указанной точностью
CURRENT_IMESTAMP (точность)	текущие дата и время с указанной точностью
EXTRACT(часть FROM значение)	указанная часть (DAY, HOUR и т.д.) из значения типа DATETIME
LOWER(строка)	строка, преобразованная к нижнему регистру
OCTETLENGTH(строка)	число байтов в строке символов
POSITION(первая строка IN вторая строка)	позиция, с которой начинается вхождение первой строки во вторую строку
SUBSTRING(строка FROM n FOR длина)	часть строки, начинающаяся с n-го символа и имеющая указанную длину
TRANSLATE(строка USING функция)	строка, преобразованная с помощью указанной функции
TRIM(BOTH символ FROM строка)	строка, в которой удалены первые и последние указанные символы
TRIM(LEADING символ FROM строка)	строка, в которой удалены первые указанные символы
TRIM(TRAILING символ FROM строка)	строка, в которой удалены последние указанные символы
UPPER(строка)	строка, преобразованная к верхнему регистру

Запросы на чтение данных. Оператор SELECT

Оператор SELECT применяется для построения выборок данных и имеет следующий синтаксис:

```
SELECT [ALL | DISTINCT] возвращаемый_столбец, … | *
FROM спецификатор_таблицы, …
WHERE условие_поиска
GROUP BY имя_столбца, …
HAVING условие_поиска
ORDER BY спецификатор_сортировки, …
```

Предложение SELECT

В предложении SELECT, с которого начинаются все операторы SELECT, необходимо указать элементы данных, которые будут возвращены в результате запроса Эти элементы задаются в виде списка возвращаемых столбцов, разделенных запятыми Для каждого элемента из этого списка в таблице результатов запроса будет создан один столбец Столбцы в таблице результатов будут расположены в том же порядке, что и элементы списка возвращаемых столбцов Возвращаемый столбец может представлять собой:

1. имя столбца, идентифицирующее один из столбцов, содержащихся в таблицах, которые перечислены в предложении FROM Когда в качестве возвращаемого столбца указывается имя столбца таблицы базы данных SQL просто берет значение этого столбца для каждой из строк таблицу и помещает его в соответствующую строку таблицы результатов запроса;

2. константа, показывающая, что в каждой строке результатов запроса должно содержаться одно и то же значение,

3. выражение, показывающее, что SQL должен вычислять значение, помещаемое в результаты запроса, по формуле, определенной в выражении.

Предложение FROM

Предложение FROM состоит из ключевого слова FROM, за которым следует список спецификаторов таблиц, разделенных запятыми Каждый спецификатор таблицы идентифицирует таблицу, содержащую данные, которые считывает запрос Такие таблицы называются исходными таблицами

запроса (и оператора SELECT), поскольку все данные, содержащиеся в таблице результатов запроса, берутся из них.

Результатом SQL-запроса на чтение всегда является таблица, содержащая данные и ничем не отличающаяся от таблиц базы данных

Кроме столбцов, значения которых считываются непосредственно из базы данных, SQL-запрос на чтение может содержать вычисляемые столбцы, значения которых определяются на основании значений, хранящихся в базе данных. Чтобы получить вычисляемый столбец, в списке возвращаемых столбцов необходимо указать выражение.

Иногда требуется получить содержимое всех столбцов таблицы. На практике такая ситуация может возникнуть, когда вы впервые сталкиваетесь с новой базой данных и необходимо быстро получить представление о ее структуре и хранимых в ней данных. С учетом этого в SQL разрешается использовать вместо списка .возвращаемых столбцов символ звездочки (*), который означает, что требуется прочитать все столбцы:

Показать все данные, содержащиеся в таблице Students.

```
SELECT *
FROM Students
```

Повторяющиеся строки из таблицы результатов запроса можно удалить, если в операторе SELECT перед списком возвращаемых столбцов указать ключевое слово DISTINCT.

Отбор строк (предложение WHERE)

SQL-запросы, считывающие из таблицы все строки, полезны при просмотре базы данных и создании отчетов, однако редко применяются для чего-нибудь еще Обычно требуется выбрать из таблицы несколько строк и включить в результаты запроса только их Чтобы указать, какие строки требуется отобрать, следует использовать предложение WHERE.

Предложение WHERE состоит из ключевого слова WHERE, за которым следует условие поиска, определяющее, какие именно строки требуется прочитать.

Если условие поиска имеет значение TRUE, строка будет включена в результаты запроса.

Если условие поиска имеет значение FALSE или NULL, то строка исключается из результатов запроса.

Условия поиска

В SQL используется множество условий поиска, позволяющих эффективно и естественным образом создавать различные типы запросов. Ниже рассматриваются пять основных условий поиска (в стандарте ANSI/ISO они называются предикатами).

Сравнение (=, <>, <, <=, >, >=). Наиболее распространенным условием поиска в SQL является сравнение. При сравнении SQL вычисляет и сравнивает значения двух выражений для каждой строки данных. Выражения могут быть как очень простыми, например содержать одно имя столбца или константу, так и более сложными – арифметическими выражениями.

Ниже приведен синтаксис оператора сравнения.

Выражение1 = | <> | < | <= | > | >= Выражение2

Проверка на принадлежность диапазону значений (BETWEEN). Другой формой условия поиска является проверка на принадлежности диапазону значений (ключевое слово BETWEEN). При этом проверяется, находится ли значение данных между двумя определенными значениями.

Проверяемое_выражение [NOT] BETWEEN нижнее_выражение AND верхнее_выражение

При проверке на принадлежность диапазону верхний и нижний пределы считаются частью диапазона, поэтому в результаты запроса.

Если выражение, определяющее нижний предел диапазона, имеет значение NULL, то проверка BETWEEN возвращает значение FALSE тогда,

когда проверяемое значение больше верхнего предела диапазона, и значение NULL в противном случае.|

Если выражение, определяющее верхний предел диапазона, имеет значение NULL, то проверка BETWEEN возвращает значение FALSE, когда проверяемое значение меньше нижнего предела, и значение NULL противном случае.

Поверка на членство в множестве (IN). Еще одним распространенным условием поиска является проверка на членство в множестве (IN). В этом случае проверяется, соответствует ли значение данных какому-либо значению из заданного списка.

проверяемое_выражение [NOT] IN (константа, …)

Например: вывести список фамилий студентов, которые учатся в группах с кодами 1, 3, 5 и 10.

```
SELECT StName
FROM Students
WHERE GrNo IN (1, 3, 5, 10)
```

Проверка на соответствие шаблону (LIKE). Для чтения строк, в которых содержимое некоторого текстового столбца совпадает с заданным текстом, можно использовать простое сравнение, однако иногда необходимо осуществить сравнение на основе фрагмента строки.

Поверка на соответствие шаблону (ключевое слово LIKE), позволяет определить, соответствует ли значение данных в столбце некоторому шаблону. Шаблон представляет собой строку, в которую может входить один или более подстановочных знаков. Эти знаки интерпретируются особым образом.

имя_столбца [NOT] LIKE шаблон [ESCAPE символ_пропуска]

Подстановочные знаки. Подстановочный знак % совпадает с любой последовательностью из нуля или более символов. Например, следующий запрос выведет информацию о студентах, чья фамилия начинается с "Иван":

SELECT *

FROM StudentsWHERE StName LIKE 'Иван%'

Подстановочный знак "_" (символ подчеркивания) совпадает с любым отдельным символом. Например, если вы уверены, что имя студентки- либо "Наталья", либо "Наталия", то можете воспользоваться следующим запросом:

SELECT *

FROM Students WHERE StName LIKE 'Натал_я'

Подстановочные знаки можно помещать в любое место строки шаблона, и в одной строке может содержаться несколько подстановочных знаков.

Символы пропуска. При проверке строк на соответствие шаблону может оказаться, что подстановочные знаки входят в строку символов в качестве литералов. Например, нельзя проверить, содержится ли знак процента в строке, просто включив его в шаблон, поскольку SQL будет считать этот знак подстановочным. Как правило, это не вызывает серьезных проблем, поскольку подстановочные знаки довольно редко встречаются в именах, названиях товаров и других текстовых данных, которые обычно хранятся в базе данных.

В стандарте ANSI/ISO определен способ проверки наличия в строках литералов, использующихся в качестве подстановочных знаков. Для этом применяются символы пропуска. Когда в шаблоне встречается такой символ то символ, следующий непосредственно за ним, считается не подстановочным знаком, а литералом. (Происходит пропуск символа.) Непосредственно за символом пропуска может следовать либо один из двух подстановочных знаков, либо сам символ пропуска, поскольку он тоже приобретает в шаблоне особое значение.

Символ пропуска определяется в виде строки, состоящей из одного символа, и предложения ESCAPE. Ниже приведен пример использования знака доллара ($) в качестве символа пропуска:

SELECT *

FROM DataTable WHERE Text LIKE '%менее 50$% %' ESCAPE $

Проверка на равенство значению NULL (IS NULL). Значения NULL обеспечивают возможность применения трехзначной логики в условиях поиска. Для любой заданной строки результат применения условия поиска может быть TRUE, FALSE или NULL (в случае, когда в одном из столбцов содержится значение NULL). Иногда бывает необходимо явно проверять значения столбцов на равенство NULL и непосредственно обрабатывать их. Для этого в SQL имеется специальная проверка на равенство значению NULL (IS NULL).

IS [NOT] NULL имя_ столбца

Составные условия поиска (AND, OR и NOT). Простые условия поиска, описанные в предыдущих параграфах, после применения к некоторой строке возвращают значения TRUE, FALSE или NULL. С помощью правил логики эти простые условия можно объединять в более сложные. Условия поиска, объединенные с помощью ключевых слов AND, OR и NOT, сами могут быть составными.

WHERE [NOT] условие_поска [AND | OR] [NOT] условие_поска …

Сортировка результатов запроса (предложение ORDER BY).

Строки результатов запроса, как и строки таблицы базы данных, не имеют определенного порядка. Включив в оператор SELECT предложение ORDER BY, можно отсортировать результаты запроса. Это предложение состоит из ключевых слов ORDER BY, за которыми следует список имен столбцов, разделенных запятыми.

ORDER BY имя_столбца [ASC | DESC], …

В предложении ORDER BY можно выбрать возрастающий или убывающий порядок сортировки. По умолчанию, данные сортируются в порядке возрастания. Чтобы сортировать их по убыванию, следует включить в предложение сортировки ключевое слово DESC.

Например: вывести список фамилий студентов учащихся в группе с кодом 1 в обратном алфавитном порядке.

```
SELECT StName
FROM Students ORDER BY DESC StName
```

Многотабличные запросы на чтение (объединения).

На практике многие запросы считывают данные сразу из нескольких таблиц базы данных.

Запросы с использованием отношения предок/потомок.

Среди многотабличных запросов наиболее распространены запросы к двум таблицам, связанным с помощью отношения предок/потомок. Запрос о студентах, учащихся в группе с некоторым названием является примером такого запроса. У каждого студента (потомка) есть соответствующая ему группа (предок), и каждая группа (предок) может иметь много студентов (потомков). Пары строк, из которых формируются результаты запроса, связаны отношением предок/потомок.

Например: вывести список фамилий студентов с названием группы, в которой он учится

```
SELECT StName, GrName
FROM Students, Groups
WHERE Students.GrNo = Groups.GrNo
```

Прочие объединения таблиц по равенству

Огромное множество многотабличных запросов основано на отношениях предок/потомок, но в SQL не требуется, чтобы связанные столбцы представляли собой пару "внешний ключ – первичный ключ". Любые два столбца из двух таблиц могут быть связанными столбцами, если только они имеют сравнимые типы данных.

Объединение таблиц по неравенству. Термин "объединение" применяется к любому запросу, который объединяет данные из двух таблиц базы данных путем сравнения значений в двух столбцах этих таблиц. Самыми распространенными являются объединения, созданные на основе равенства связанных столбцов (объединения по равенству). Кроме того, SQL позволяет объединять таблицы с помощью других операций сравнения. Например, получить все коды дисциплин по которым студентом с кодом 1 была получена оценка большая, чем по дисциплине с кодом 1.

```
SELECT Marks1.SubjNo
FROM Marks AS Marks1, Marks AS Marks2
WHERE Marks1.Mark > Marks2.Mark
AND Marks2.SubjNo = 1 AND Marks1.StNo = 1
```

Следует признать, что данный пример имеет несколько искусственный характер и является иллюстрацией того, почему столь мало распространены объединения по неравенству. Однако они могут оказаться полезными в приложениях, предназначенных для поддержки принятия решений, и в других приложениях, исследующих более сложные взаимосвязи в базе данных.

Полные имена столбцов. В учебной базе данных имеется несколько случаев, когда две таблицы содержат столбцы с одинаковыми именами. Например, столбцы с именем GrNo имеются в таблицах Groups и Students.

Чтобы исключить разночтения, при указании столбцов необходимо использовать их полные имена. Полное имя столбца содержит имя столбца и имя таблицы, в которой он находится. Полные имена двух столбцов GrNo в учебной базе данных будут такими:

```
Groups.GrNo    Students.GrNo
```

В операторе SELECT вместо простых имен столбцов всегда можно использовать полные имена. Таблица, заданная в полном имени столбца, должна, конечно, соответствовать одной из таблиц, заданных в предложении FROM.

Чтение всех столбцов. Как уже говорилось ранее, оператор SELECT * используется для чтения всех столбцов таблицы, указанной в предложении FROM. В многотабличном запросе звездочка означает выбор всех столбцов из всех таблиц, указанных в предложении FROM.

Самообъединения. Некоторые многотабличные запросы используют отношения, существующие внутри одной из таблиц. Предположим, например, что требуется вывести список имен всех преподавателей и их руководителей. Каждому преподавателю соответствует одна строка в таблице Teachers, а столбец ChiefNo содержит идентификатор преподавателя, являющегося руководителем. Столбцу ChiefNo следовало бы быть внешним ключом для таблицы, в которой хранятся данные о руководителях. И он им, фактически, является – это внешний ключ для самой таблицы Teachers.

Для объединения таблицы с самой собой в SQL применяется именно такой подход: использование "виртуальной копии". Вместо того чтобы на самом деле сделать копию таблицы, SQL позволяет вам сослаться на нее, используя другое имя, которое называется псевдонимом таблицы.

Например: вывести список всех преподавателей и их руководителей.

```
SELECT Teachers. TName, Chiefs.TName
FROM Teachers, Teachers Chiefs
WHERE Teachers. ChiefNo = Chiefs.TNo
```

Псевдонимы таблиц. Как уже было сказано в предыдущем параграфе, псевдонимы таблиц необходимы в запросах, включающих самообъединения. Однако псевдоним можно использовать в любом запросе (например, если запрос касается таблицы другого пользователя или если имя таблицы очень длинное и использовать его в полных именах столбцов утомительно).

Внешнее объединение таблиц. Операция объединения в SQL соединяет информацию из двух таблиц, формируя пары связанных строк из этих двух таблиц. Объединенную таблицу образуют пары тех строк из различных таблиц, у которых в связанных столбцах содержатся одинаковые значения. Если строка одной из таблиц не имеет пары, то такой вид объединения, называемый

внутренним объединением, может привести к неожиданным результатам (потере некоторых данных в результате запроса). Для создания объединений таблиц, имеющих неодинаковые значения в столбцах, на основе которых осуществляется связь, применяют внешнее объединение таблиц наиболее полно представленное в стандарте SQL2.

Объединения и стандарт SQL2

В стандарте SQL2 был определен совершенно новый метод поддержки внешних объединений, который не опирался ни на одну популярную СУБД. В спецификации стандарта SQL2 поддержка внешних объединений осуществлялась в предложении FROM с тщательно разработанным синтаксисом, позволявшим пользователю точно определить, как исходные таблицы должны быть объединены в запросе. Расширенное предложение FROM поддерживает также операцию UNION над таблицами и допускает сложные комбинации запросов на объединение операторов SELECT и объединений таблиц.

Внутренние объединения в стандарте SQL2

Две объединяемые таблицы соединяются явно посредством операции JOIN, а условие поиска, описывающее объединение, находится теперь в предложении ON внутри предложения FROM В условии поиска, следующем за ключевым словом ON, могут быть заданы любые критерии сравнения строк двух объединяемых таблиц.

Например: вывести список фамилий студентов, и названия групп, к которых они учаться.

```
SELECT StName, GrName
FROM Students INNER JOIN Groups
ON Students.GrNo = Groups.GrNo
```

(В этих простых двухтабличных объединениях все содержимое предложения WHERE просто перешло в предложение ON, и предложение ON не добавляет ничего нового в язык SQL.

Стандарт SQL2 допускает еще один вариант запроса на простое внутреннее объединение таблиц Students и Groups. Так как связанные столбцы этих таблиц имеют одинаковые имена и сравниваются на предмет равенства (что делается довольно часто), то можно использовать альтернативную форму предложения ON, в которой задается список имен связанных столбцов:

```
SELECT StName, GrName
FROM Students INNER JOIN Groups USING (GrNo)
```

Ниже приведен синтаксис оператора JOIN:

1. естественное соединение.

```
FROM спецификация_таблиц,…,
таблица1
NATURAL {INNER|FULL[OUTER]|LEFT[OUTER]|RIGHT[OUTER]}JOIN
таблица2 …
```

2. соединение с использованием выражения.

```
FROM спецификация_таблицы,…,
таблица1
{INNER|[OUTER] FULL|[OUTER]LEFT|[OUTER]RIGHT} JOIN
таблица2
ON условие | USING(список_столбцов),…
```

3. объединение или декартово произведение.

```
FROM спецификация_таблицы,…,
таблица1 {UNION | CROSS JOIN} таблица2 ,…
```

Объединение двух таблиц, в котором связанные столбцы имеют идентичные имена, называется естественным объединением, так как обычно это действительно самый "естественный" способ объединения двух таблиц. Запрос на выборку пар фамилия студента/название группы, в которой он учиться, можно выразить как естественное объединение следующим образом:

```
SELECT StName, GrName
FROM Students NATURAL INNER JOIN Groups
```

Внешние объединения в стандарте SQL2

Стандарт SQL2 обеспечивает полную поддержку внешних объединений, расширяя языковые конструкции, используемые для внутренних объединений. Например, для построения таблицы подчиненности преподавателей можно применить следующий запрос:

```
SELECT Chief.TName, SubOrdinate.TName
FROM Teachers AS Chief FULL OUTER JOIN Teachers AS SubOrdinate
ON Chief.TNo = SubOrdinate.TChiefNo
```

Результат такого запроса (данные из Приложения А) приведен на рис.

Chief.TName	SubOrdinate.TName
NULL	Иванов
Иванов	Петров
Петров	Стрельцов
Петров	Сидоров
Сидоров	NULL
Стрельцов	NULL

Результатом такого запроса на внешнее объединение.

Таблица результатов запроса будет содержать по одной строке для каждой связанной пары начальник/подчиненный, а также по одной строке для каждой несвязанной записи для начальника или подчиненного, расширенной значениями NULL в столбцах другой таблицы.

Ключевое слово OUTER, так же как и ключевое слово INNER, в стандарте SQL2 не является обязательным. Поэтому предыдущий запрос можно, было бы переписать следующим образом:

```
SELECT Chief.TName, SubOrdinate.TName
FROM Teachers AS Chief FULL JOIN Teachers AS SubOrdinate
ON Chief.TNo = SubOrdinate.TChiefNo
```

По слову FULL СУБД сама определяет, что запрашивается внешнее объединение.

Вполне естественно, что в стандарте SQL2 левое и правое внешние объединения обозначаются словами LEFT и RIGHT вместо слова FULL. Вот вариант того же запроса, определяющий левое внешнее объединение:

SELECT Chief.TName, SubOrdinate.TName

FROM Teachers AS Chief LEFT OUTER JOIN Teachers AS SubOrdinate

ON Chief.TNo = SubOrdinate.TChiefNo

В результате такого запроса (данные из Приложения А.) будет получено следующее отношение.

Chief.TName	SubOrdinate.TName
Иванов	Петров
Петров	Стрельцов
Петров	Сидоров
Сидоров	NULL
Стрельцов	NULL

Результатом такого запроса на внешнее объединение

Перекрестные объединения и запросы на объединение в SQL2

Расширенное предложение FROM в стандарте SQL2 поддерживает также два других способа соединения данных из двух таблиц – декартово произведение и запросы на объединение. Строго говоря, ни один из них не является операцией "объединения", но они поддерживаются в стандарте SQL2 с помощью тех же самых предложений, что и внутренние и внешние объединения. Вот запрос, создающий декартово произведение таблиц Students и Groups:

SELECT *

FROM Students CROSS JOIN Groups

Многотабличные объединения в стандарте SQL2

Одно из крупных преимуществ расширенного предложения FROM заключается в том, что оно дает единый стандарт для определения как внутренних и внешних объединений, так и произведений и запросов на объединение. Другим, даже еще более важным преимуществом этого предложения является то, что оно обеспечивает очень ясную и четкую спецификацию объединений трех и четырех таблиц, а также произведений и запросов на объединение. Для построения этих сложных объединений любые

выражения описанные ранее, могут быть заключены в круглые скобки. Результирующее выражение, в свою очередь, можно использовать для создания других выражений объединения, как если бы оно было простой таблицей. Точно так же, как SQL позволяет с помощью круглых скобок комбинировать различные арифметические операции (+, –, * и /) и строить сложные выражения, стандарт SQL2 дает возможность создавать сложные выражения для объединений.

Итоговые запросы на чтение. Агрегатные функции

Для подведения итогов по информации, содержащейся в базе данных, в SQL предусмотрены агрегатные (статистические) функции. Агрегатная функция принимает в качестве аргумента какой-либо столбец данных целиком, а возвращает одно значение, которое определенным образом подытоживает этот столбец. Например, агрегатная функция AVG() принимает в качестве аргумента столбец чисел и вычисляет их среднее значение.

В SQL имеется шесть агрегатных функций, которые позволяют получать различные виды итоговой информации. Ниже описан синтаксис этих функций:

1. функция SUM() вычисляет сумму всех значений, содержащихся в столбце:

SUM(выражение | [DISTINCT] имя_столбца)

2. функция AVG() вычисляет среднее всех значений, содержащихся в столбце:

AVG(выражение | [DISTINCT] имя_столбца)

3. функция MIN() находит наименьшее среди всех значений, содержащихся в столбце:

MIN(выражение | имя_столбца)

4. функция MAX() находит наибольшее среди всех значений, содержащихся в столбце:

MAX(выражение | имя_столбца)

5. функция COUNT() подсчитывает количество значений, содержащихся в столбце:

COUNT([DISTINCT] имя_столбца)

6. функция COUNT(*) подсчитывает количество строк в таблице результатов запроса:

COUNT(*)

Агрегатные функции и значения NULL

В стандарте ANSI/ISO также определены следующие точные правила обработки значений NULL в агрегатных функциях:

1. если какие-либо из значений, содержащихся в столбце, равны NULL, при вычислении результата функции они исключаются;

2. если все значения в столбце равны NULL, то функции SUM(), AVG(), MIN() и MAX () возвращают значение NULL; функция COUNT () возвращает ноль;

3. если в столбце нет значений (т.е. столбец пустой), то функции SUM() , AVG(), MIN() и MAX() возвращают значение NULL; функция COUNT() возвращает ноль;

4. функция COUNT(*) подсчитывает количество строк и не зависит от наличия или отсутствия в столбце значений NULL; если строк в таблице нет, эта функция возвращает ноль.

Запросы с группировкой (предложение GROUP BY)

Итоговые запросы, о которых до сих пор шла речь в настоящей главе рассчитывают итоговые результаты на основании всех записей в таблице. Однако необходимость в таких вычислениях возникает достаточно редко, чаще бывает необходимо получать промежуточные итоги на основании групп записей в таблице. Эту возможность предоставляет предложение GROUP BY оператора SELECT.

Запрос, включающий в себя предложение GROUP BY, называется запросом с группировкой, поскольку он объединяет строки исходных таблиц в группы и для каждой группы строк генерирует одну строку таблицы результатов запроса. Столбцы, указанные в предложении GROUP BY, называются столбцами группировки, поскольку именно они определяют, по

какому признаку строки делятся на группы. Например, получить список фамилий студентов и их средних оценок.

```
SELECT StName, AVG(Mark)
FROM Marks INNER JOIN Students USING(StNo)
GROUP BY StName
```

Несколько столбцов группировки

SQL позволяет группировать результаты запроса на основании двух или более столбцов. Например, получить список фамилий студентов и их средних оценок за каждый семестр.

```
SELECT StName, Semester, AVG(Mark)
FROM Marks INNER JOIN Students USING(StNo)
GROUP BY StName, Semester
```

Ограничения на запросы с группировкой

На запросы, в которых используется группировка, накладываются дополнительные ограничения. Столбцы с группировкой должны представлять собой реальные столбцы таблиц, перечисленных в предложении FROM. Нельзя группировать строки на основании значения вычисляемого выражения.

Кроме того, существуют ограничения на элементы списка возвращаемых столбцов. Все элементы этого списка должны иметь одно значение для каждой группы строк. Это означает, что возвращаемым столбцом может быть:

1. константа;
2. агрегатная функция, возвращающая одно значение для всех строк, входящих в группу;
3. столбец группировки, который, по определению, имеет одно и то же значение во всех строках группы;
4. выражение, включающее в себя перечисленные выше элементы.

На практике в список возвращаемых столбцов запроса с группировкой всегда входят столбец группировки и агрегатная функция. Если последняя не указана, значит, запрос можно более просто выразить с помощью ключевого слова DISTINCT без использования предложения GROUP BY. И наоборот, если

не включить в результаты запроса столбец группировки, вы не сможете определить, к какой группе относится каждая строка результатов.

Значения NULL в столбцах группировки

В стандарте ANSI определено, что два значения NULL в предложении GROUP BY равны.

Строки, имеющие значение NULL в одинаковых столбцах группировки и идентичные значения во всех остальных столбцах группировки, помещаются в одну группу.

Условия поиска групп (предложение HAVING)

Точно так же, как предложение WHERE используется для отбора отдельных строк, участвующих в запросе, предложение HAVING можно применить для отбора групп строк. Его формат соответствует формату предложения WHERE. Предложение HAVING состоит из ключевого слова HAVING, за которым следует условие поиска. Таким образом, данное предложение определяет условие поиска для групп.

Ограничения на условия поиска групп

Предложение HAVING используется для того, чтобы включать и исключать группы строк из результатов запроса, поэтому на используемое в нем условие поиска наложены такие же ограничения, как и на элементы списка возвращаемых столбцов.

Предложение HAVING без GROUP BY

Предложение HAVING почти всегда используется в сочетании с предложением GROUP BY, однако синтаксис оператора SELECT не требует этого. Если предложение HAVING используется без предложения GROUP BY, SQL рассматривает полные результаты запроса как одну группу. Другими словами, агрегатные функции, содержащиеся в предложении HAVING, применяются к одной и только одной группе, и эта группа состоит из всех строк. На практике предложение HAVING очень редко используется без соответствующего предложения GROUP BY.

Вложенные запросы

Вложенным (или подчиненным) запросом называется запрос, содержащийся в предложении WHERE или HAVING другого оператора SQL. Вложенные запросы позволяют естественным образом обрабатывать запросы, выраженные через результаты других запросов.

Чаще всего вложенные запросы указываются в предложении WHERE оператора SQL. Когда вложенный запрос содержится в данном предложении, он участвует в процессе отбора строк.

Например, вывести список фамилий студентов, средний балл которых выше 4,5:

```
SELECT StName
FROM Students
WHERE (SELECT AVG(Mark)
   FROM Marks
   WHERE Marks.StNo = Students.StNo) > 4.5
```

Вложенный запрос всегда заключается в круглые скобки, но по-прежнему сохраняет знакомую структуру оператора SELECT, содержащего предложение FROM и необязательные предложения WHERE, GROUP BY и HAVING. Структура этих предложений во вложенном запросе идентична их структуре в оператора SELECT; во вложенном запросе эти предложения выполняют свои обычные функции. Однако между вложенным запросом и оператором SELECT имеется ряд отличий:

1. Таблица результатов вложенного запроса всегда состоит из одного столбца. Это означает, что в предложении SELECT вложенного запроса всегда указывается один возвращаемый столбец.

2. Во вложенный запрос не может входить предложение ORDER BY. Результаты вложенного запроса используются только внутри главного запроса и для пользователя остаются невидимыми, поэтому нет смысла их сортировать.

3. Вложенный запрос не может быть запросом на объединение нескольких различных операторов SELECT; допускается использование только одного оператора SELECT.

4. Имена столбцов во вложенном запросе могут являться ссылками на столбцы таблиц главного запроса. Это так называемые внешние ссылки (ссылка на Students.StNo в предыдущем примере). Внешние ссылки необязательно должны присутствовать во вложенном запросе.

Условия поиска во вложенном запросе

Вложенный запрос всегда является частью условия поиска в предложении WHERE или HAVING. Ранее были рассмотрены простые условия поиска, которые могут использоваться в этих предложениях. Кроме того, в SQL имеются следующие условия поиска во вложенном запросе.

Сравнение с результатом вложенного запроса (=, <>, <, <=, >, >=). Сравнивает значение выражения с одним значением, возвращенным вложенным запросом. Эта проверка напоминает простое сравнение.

проверяемое_выражение | = | <> | < | <= | > | >= | вложенный_запрос

Сравнение с результатом вложенного запроса является модифицированной формой простого сравнения. Значение выражения сравнивается со значением, возвращенным вложенным запросом, и если условие сравнения выполняется, то проверка возвращает значение TRUE. Эта проверка используется для сравнения значения из проверяемой строки с одним значением, возвращенным вложенным запросом, как показано первом примере.

Проверка на принадлежность результатам вложенного запроса (IN). Проверка на принадлежность результатам вложенного запроса (ключевое слово IN) является видоизмененной формой простой проверки на членство в множестве.

проверяемое_выражение [NOT] IN вложенный_запрос

Одно значение сравнивается со столбцом данных, возвращенных вложенным запросом, и если это значение равно одному из значений в столбце, проверка возвращает TRUE. Данная проверка используется, когда необходимо

сравнить значение из проверяемой строки с множеством значений, возвращенных вложенным запросом. Например, вывести всю информацию о студентах, учащихся в группах сназваниями, начинающимися на букву ″А″:

SELECT *

FROM Students

WHERE GrNo IN (SELECT GrNo FROM Groups WHERE GrName LIKE ‘A%’)

Проверка на существование (EXISTS). В результате проверки на существование (ключевое слово EXISTS) можно выяснить, содержится ли в таблице результатов вложенного запроса хотя бы одна строка. Аналогичной простой проверки не существует. Проверка на существование используется только с вложенными запросами.

[NOT] EXISTS вложенный_запрос

Например, вывести фамилии студентов, которые в 1-м семестре сдали хотябы одну дисциплину:

SELECT StName

FROM Students

WHERE EXISTS (SELECT *

FROM Marks

WHERE Marks.StNo = Students.StNo AND Semester = 1)

Многократное сравнение (ANY и ALL). В проверке IN выясняется, не равно ли некоторое значение одному из значений, содержащихся в столбце результатов вложенного запроса. В SQL имеется две разновидности многократного сравнения – ANY и ALL, расширяющие предыдущую проверку до уровня других операторов сравнения. В обеих проверках некоторое значение сравнивается со столбцом данных, возвращенным вложенным запросом.

проверяемое_выражение | = | <> | < | <= | > | >= | ANY | ALL вложенный_запрос

Проверка ANY. В проверке ANY используется один из шести операторов сравнения (=, <>, <, <=, >, >=) для того, чтобы сравнить одно проверяемое

значение со столбцом данных, возвращенным вложенным запросом. Проверяемое значение поочередно сравнивается с каждым значением, содержащимся в столбце. Если любое из этих сравнений дает результат TRUE, то проверка ANY возвращает значение TRUE.

Например, вывести список фамилий студентов, получивших в первом семестре хотя бы одну отличную оценку.

```
SELECT StName
FROM Students
WHERE 5 = ANY (SELECT Mark
   FROM Marks
   WHERE Marks.StNo = Students.StNo AND Semester = 1)
```

Если вложенный запрос в проверке ANY не создает ни одной строки результата или если результаты содержат значения NULL, то в различных СУБД проверка ANY может выполняться по-разному. В стандарте ANSI/ISO для языка SQL содержатся подробные правила, определяющие результаты проверки ANY, когда проверяемое значение сравнивается со столбцом результатов вложенного запроса:

1. если вложенный запрос возвращает пустой столбец результатов, то проверка ANY имеет значение FALSE (в результате выполнения вложенного запроса не получено ни одного значения, для которого выполнялось бы условие сравнения).

2. если операция сравнения имеет значение TRUE хотя бы для одного значения в столбце, то проверка ANY возвращает значение TRUE (имеется некоторое значение, полученное вложенным запросом, для которого условие сравнения выполняется).

3. если операция сравнения имеет значение FALSE для всех значений в столбце, то проверка ANY возвращает значение FALSE (можно утверждать, что ни для одного значения, возвращенного вложенным запросом, условие сравнения не выполняется);

4. если операция сравнения не имеет значение TRUE ни для одного значения в столбце, но в нем имеется одно или несколько значений NULL то проверка ANY возвращает результат NULL. (В этой ситуации невозможно но с определенностью утверждать, существует ли полученное вложенным запросом значение, для которого выполняется условие сравнения; может быть, существует, а может и нет – все зависит от "настоящих" значений неизвестных данных.)

Проверка ALL. В проверке ALL, как и в проверке ANY, используется один из шести операторов (=, <>, <, <=, >, >=) для сравнения одного проверяемого значения со столбцом данных, возвращенным вложенным запросом. Проверяемое значение поочередно сравнивается с каждым значением, содержащимся в столбце. Если все сравнения дают результат TRUE, то проверка ALL возвращает значение TRUE.

Например, вывести список фамилий студентов, получивших в первом семестре только удовлетворительные оценки.

```
SELECT StName
FROM Students
WHERE 3 = ALL (SELECT Mark
   FROM Marks
   WHERE Marks.StNo = Students.StNo AND Semester = 1)
```

Если вложенный запрос в проверке ALL не возвращает ни одной строки или если результаты запроса содержат значения NULL, то в различных СУБД проверка ALL может выполняться по-разному. В стандарте ANSI/ISO для языка SQL содержатся подробные правила, определяющие результаты проверки ALL, когда проверяемое значение сравнивается со столбцом результатов вложенного запроса:

1. если вложенный запрос возвращает пустой столбец результатов, то проверка ALL имеет значение TRUE. Считается, что условие сравнения выполняется, даже если результаты вложенного запроса отсутствуют.

2. если операция сравнения дает результат TRUE для каждого значения в столбце, то проверка ALL возвращает значение TRUE. Условие сравнения выполняется для каждого значения, полученного вложенным запросом.

3. если операция сравнения дает результат FALSE для какого-нибудь значения в столбце, то проверка ALL возвращает значение FALSE. В этом, случае можно утверждать, что условие поиска выполняется не для каждого значения, полученного вложенным запросом.

4. если операция сравнения не дает результат FALSE ни для одного значения в столбце, но для одного или нескольких значений дает результат NULL, то проверка ALL возвращает значение NULL. В этой ситуации нельзя с определенностью утверждать, для всех ли значений, полученных вложенным запросом, справедливо условие сравнения; может быть, для всех, а может и нет – все зависит от "настоящих" значений неизвестных данных.

Вложенные запросы и объединения

При чтении данной главы вы, возможно, заметили, что многие запросы, записанные с применением вложенных запросов, можно также записать в виде многотабличных запросов. Такое случается довольно часто, и SQL позволяет записать запрос любым способом.

Уровни вложенности запросов

Все рассмотренные до сих пор запросы были "двухуровневыми" и состояли из главного и вложенного запросов. Точно так же, как внутри главной запроса может находится вложенный запрос, внутри вложенного запроси может находиться еще один вложенный запрос.

Вложенные запросы в предложении HAVING

Хотя вложенные запросы чаще всего применяются в предложении WHERE их можно использовать и в предложении HAVING главного запроса. Когда вложенный запрос содержится в предложении HAVING, он участвует в отбор группы строк.

Тема 8. ОБЕСПЕЧЕНИЕ БЕЗОПАСНОСТИ БД

Термины безопасность и целостность в контексте обсуждения баз данных часто используется совместно, хотя на самом деле, это совершенно разные понятия. Термин безопасность относится к защите данных от несанкционированного доступа, изменения или разрушения данных, а целостность – к точности или истинности данных. По-другому их можно описать следующим образом:

5. под безопасностью подразумевается, что пользователям разрешается выполнять некоторые действия;

6. под целостностью подразумевается, что эти действия выполняются корректно.

Между ними есть, конечно, некоторое сходство, поскольку как при обеспечении безопасности, так и при обеспечении целостности система вынуждена проверить, не нарушают ли выполняемые пользователем действия некоторых правил. Эти правила должны быть заданы (обычно администратором базы данных) на некотором удобном для этого языке и сохранены в системном каталоге. Причем в обоих случаях СУБД должна каким-то образом отслеживать все действия пользователя и проверять их соответствие заданным правилам.

Среди многочисленных аспектов проблемы безопасности необходимо отметить следующие:

7. Правовые, общественные и этические аспекты (имеет ли право некоторое лицо получить запрашиваемую информацию, например об оценках студента).

8. Физические условия (например, закрыт ли данный компьютер или терминальная комната или защищен каким-либо другим образом).

9. Организационные вопросы (например, как в рамках предприятия, обладающего некой системой, организован доступ к данным).

10. Вопросы реализации управления (например, если используется метод доступа по паролю, то как организована реализация управления и как часто меняются пароли).

11. Аппаратное обеспечение (обеспечиваются ли меры безопасности на аппаратном уровне, например, с помощью защитных ключей или привилегированного режима управления).

12. Безопасность операционной системы (например, затирает ли базовая операционная система содержание структуры хранения и файлов с данными при прекращении работы с ними).

13. И наконец, некоторые вопросы, касающиеся непосредственно самой системы управления базами данных (например, существует ли для базы данных некоторая концепция предоставления прав владения данными).

В настоящей лекции рассматриваются вопросы, касающиеся последнего пункта этого перечня.

Методы обеспечения безопасности

В современных СУБД поддерживается один из двух широко распространенных подходов к вопросу обеспечения безопасности данных, а именно избирательный подход или обязательный подход. В обоих подходах единицей данных или "объектом данных", для которых должна быть создана система безопасности, может быть как вся база данных целиком или какой-либо набор отношений, так и некоторое значение данных для заданного атрибута внутри некоторого кортежа в определенном отношении. Эти подходы отличаются следующими свойствами:

14. В случае избирательного управления некий пользователь обладает различными правами (привилегиями или полномочиями) при работе с разными объектами. Более того, разные пользователи обычно обладают и разными правами доступа к одному и тому же объекту. Поэтому избирательные схемы характеризуются значительной гибкостью.

15. В случае обязательного управления, наоборот, каждому объекту данных присваивается некоторый классификационный уровень, а каждый

пользователь обладает некоторым уровнем допуска. Следовательно, при таком подходе доступом к определенному объекту данных обладают только пользователи с соответствующим уровнем допуска. Поэтому обязательные схемы достаточно жестки и статичны.

Независимо от того, какие схемы используются – избирательные или обязательные, все решения относительно допуска пользователей к выполнению тех или иных операций принимаются на стратегическом, а не техническом уровне. Поэтому они находятся за пределами досягаемости самой СУБД, и все, что может в такой ситуации сделать СУБД, – это только привести в действие уже принятые ранее решения. Исходя из этого, можно отметить следующее:

Во-первых. Результаты стратегических решений должны быть известны системе (т.е. выполнены на основе утверждений, заданных с помощью некоторого подходящего языка) и сохраняться в ней (путем сохранения их в каталоге в виде правил безопасности, которые также называются полномочиями).

Во-вторых. Очевидно, должны быть некоторые средства регулирования запросов доступа по отношению к соответствующим правилам безопасности. (Здесь под "запросом, доступа" подразумевается комбинация запрашиваемой операции, запрашиваемого, объекта и запрашивающего пользователя.) Такая проверка выполняется подсистемой безопасности СУБД, которая также называется подсистемой полномочий.

В-третьих. Для того чтобы разобраться, какие правила безопасности к каким запросам доступа применяются, в системе должны быть предусмотрены способы опознания источника этого запроса, т.е. опознания запрашивающего пользователя. Поэтому в момент входа в систему от пользователя обычно требуется ввести не только его идентификатор (например, имя или должность), но также и пароль (чтобы подтвердить свои права на заявленные ранее идентификационные данные). Обычно предполагается, что пароль известен только системе и некоторым лицам с особыми правами.

В отношении последнего пункта стоит заметить, что разные пользователи могут обладать одним и тем же идентификатором некоторой группы. Таким образом, в системе могут поддерживаться группы пользователей и обеспечиваться одинаковые права доступа для пользователей одной группы, например для всех лиц из расчетного отдела. Кроме того, операции добавления отдельных пользователей в группу или их удаления из нее могут выполняться независимо от операции задания привилегий для этой группы. Обратите внимание, однако, что местом хранения информации о принадлежности к группе также является системный каталог (или, возможно, база данных).

Перечисленные выше методы управления доступом на самом деле являются частью более общей классификации уровней безопасности. Прежде всего в этих документах определяется четыре класса безопасности (security classes) – D, C, B и A. Среди них класс D наименее безопасный, класс C – более безопасный, чем класс D, и т.д. Класс D обеспечивает минимальную защиту, класс C – избирательную, класс B – обязательную, а класс A – проверенную защиту.

Избирательная защита. Класс C делится на два подкласса – C1 и C2 (где подкласс C1 менее безопасен, чем подкласс C2), которые поддерживают избирательное управление доступом в том смысле, что управление доступом осуществляется по усмотрению владельца данных.

Согласно требованиям класса C1 необходимо разделение данных и пользователя, т.е. наряду с поддержкой концепции взаимного доступа к данным здесь возможно также организовать раздельное использование данных пользователями.

Согласно требованиям класса C2 необходимо дополнительно организовать учет на основе процедур входа в систему, аудита и изоляции ресурсов.

Обязательная защита. Класс B содержит требования к методам обязательного управления доступом и делится на три подкласса – B1, B2 и B3 (где B1 является наименее, а B3 – наиболее безопасным подклассом).

Согласно требованиям класса B1 необходимо обеспечить "отмеченную защиту" (это значит, что каждый объект данных должен содержать отметку о его уровне классификации, например: секретно, для служебного пользования и т.д.), а также неформальное сообщение о действующей стратегии безопасности.

Согласно требованиям класса B2 необходимо дополнительно обеспечить формальное утверждение о действующей стратегии безопасности, а также обнаружить и исключить плохо защищенные каналы передачи информации.

Согласно требованиям класса B3 необходимо дополнительно обеспечить поддержку аудита и восстановления данных, а также назначение администратора режима безопасности.

Проверенная защита. Класс A является наиболее безопасным и согласно его требованиям необходимо математическое доказательство того, что данный метод обеспечения безопасности совместимый и адекватен заданной стратегии безопасности.

Хотя некоторые коммерческие СУБД обеспечивают обязательную защиту на уровне класса B1, обычно они обеспечивают избирательное управление на уровне класса C2.

Избирательное управление доступом

Избирательное управление доступом поддерживается многими СУБД. Избирательное управление доступом поддерживается в языке SQL.

В общем случае система безопасности таких СУБД базируется на трех компонентах:

16. Пользователи. СУБД выполняет любое действия с БД от имени какого-то пользователя. Каждому пользователю присваивается идентификатор – короткое имя, однозначно определяющее пользователя в СУБД. Для подтверждения того, что пользователь может работать с введенным идентификатором используется пароль. Таким образом, с помощью идентификатора и пароля производится идентификация и аутентификация пользователя. Большинство коммерческих СУБД позволяет объединять

пользователей с одинаковыми привилегиями в группы – это позволяет упростить процесс администрирования.

17. Объекты БД. По стандарту SQL2 защищаемыми объектами в БД являются таблицы, представления, домены и определенные пользователем наборы символов. Большинство коммерческих СУБД расширяет список объектов, добавляя в него хранимые процедуры и др. объекты.

18. Привилегии. Привилегии показывают набор действий, которые возможно производить над тем или иным объектом. Например пользователь имеет привилегию для просмотра таблицы.

Обязательное управление доступом

Методы обязательного управления доступом применяются к базам данных, в которых данные имеют достаточно статичную или жесткую структуру, свойственную, например, правительственным или военным организациям. Как уже отмечалось, основная идея заключается в том, что каждый объект данных имеет некоторый уровень классификации, например: секретно, совершенно секретно, для служебного пользования и т.д., а каждый пользователь имеет уровень допуска с такими же градациями, что и в уровне классификации. Предполагается, что эти уровни образуют строгий иерархический порядок, например: совершенно секретно $\rightarrow$ секретно $\rightarrow$ для служебного пользования и т.д. Тогда на основе этих сведений можно сформулировать два очень простых правила безопасности:

19. пользователь имеет доступ к объекту, только если его уровень допуска больше или равен уровню классификации объекта.

20. пользователь может модифицировать объекту, только если его уровень допуска равен уровню классификации объекта.

Правило 1 достаточно очевидно, а правило 2 требует дополнительных разъяснений. Прежде всего следует отметить, что по-другому второе правило можно сформулировать так: любая информация, записанная некоторым пользователем, автоматически приобретает уровень, равный уровню классификации этого пользователя. Такое правило необходимо, например, для

того, чтобы предотвратить запись секретных данных, выполняемую пользователем с уровнем допуска "секретно", в файл с меньшим уровнем классификации, что нарушает всю систему секретности.

В последнее время методы обязательного управления доступом получили широкое распространение. Требования к такому управлению доступом изложены в двух документах, которые неформально называются "оранжевой" книгой (Orange Book) и "розовой" книгой (Lavender Book). В "оранжевой" книге перечислен набор требований к безопасности для некой "надежной вычислительной базы" (Trusted Computing Base), а в "розовой" книге дается интерпретация этих требований для систем управления базами данных.

Шифрование данных

До сих пор в этой главе подразумевалось, что предполагаемый нелегальный пользователь пытается незаконно проникнуть в базу данных с помощью обычных средств доступа, имеющихся в системе. Теперь следует рассмотреть случай, когда такой пользователь пытается проникнуть в базу данных, минуя систему, т.е. физически перемещая часть базы данных или подключаясь к коммуникационному каналу. Наиболее эффективным методом борьбы с такими угрозами является шифрование данных, т.е. хранение и передача особо важных данных в зашифрованном виде.

Для обсуждения основных концепций кодирования данных следует ввести некоторые новые понятия. Исходные (незакодированные) данные называются открытым текстом. Открытый текст шифруется с помощью специального алгоритма шифрования. В качестве входных данных для такого алгоритма выступают открытый текст и ключ шифрования, а в качестве выходных – зашифрованная форма открытого текста, которая называется зашифрованным текстом. Если детали алгоритма шифрования могут быть опубликованы или, по крайней мере, могут не утаиваться, то ключ шифрования обязательно хранится в секрете. Именно зашифрованный текст, который непонятен тем, кто не обладает ключом шифрования, хранится в базе данных и передается по коммуникационному каналу.

Контрольный след выполняемых операций

Важно понимать, что не бывает неуязвимых систем безопасности, поскольку настойчивый потенциальный нарушитель всегда сможет найти способ преодоления всех систем контроля, особенно если за это будет предложено достаточно высокое вознаграждение. Поэтому при работе с очень важными данными или при выполнении критических операций возникает необходимость регистрации контрольного следа выполняемых операций. Если, например, противоречивость данных приводит к подозрению, что совершено несанкционированное вмешательство в базу данных, то контрольный след должен быть использован для прояснения ситуации и подтверждения того, что все процессы находятся под контролем. Если это не так, то контрольный след поможет, по крайней мере, обнаружить нарушителя.

Для сохранения контрольного следа обычно используется особый файл, в котором система автоматически записывает все выполненные пользователями операции при работе с обычной базой данных. Типичная запись в файле контрольного следа может содержать такую информацию:

21. запрос (исходный текст запроса);
22. терминал, с которого была вызвана операция;
23. пользователь, задавший операцию;
24. дата и время запуска операции;
25. вовлеченные в процесс исполнения операции базовые отношения, кортежи и атрибуты;
26. старые значения;
27. новые значения.

Как уже упоминалось ранее, даже констатация факта, что в данной системе поддерживается контрольное слежение, в некоторых случаях весьма существенна для предотвращения несанкционированного проникновения в систему.

Поддержка мер обеспечения безопасности в языке SQL

В действующем стандарте языка SQL предусматривается поддержка только избирательного управления доступом. Она основана на двух более или менее независимых частях SQL. Одна из них называется механизмом представлений, который (как говорилось выше) может быть использован для скрытия очень важных данных от несанкционированных пользователей. Другая называется подсистемой полномочий и наделяет одних пользователей правом избирательно и динамично задавать различные полномочия другим пользователям, а также отбирать такие полномочия в случае необходимости.

Директивы GRANT и REVOKE

Механизм представлений языка SQL позволяет различными способами разделить базу данных на части таким образом, чтобы некоторая информация была скрыта от пользователей, которые не имеют прав для доступа к ней. Однако этот режим задается не с помощью параметров операций, на основе которых санкционированные пользователи выполняют те или иные действия с заданной частью данных. Эта функция (как было показано выше) выполняется с помощью директивы GRANT.

Обратите внимание, что создателю любого объекта автоматически предоставляются все привилегии в отношении этого объекта.

Стандарт SQL1 определяет следующие привилегии для таблиц:

28. SELECT – позволяет считывать данные из таблицы или представления;
29. INSERT – позволяет вставлять новые записи в таблицу или представление;
30. UPDATE – позволяет модифицировать записи из таблицы или представления;
31. DELETE – позволяет удалять записи из таблицы или представления.

Стандарт SQL2 расширил список привилегий для таблиц и представлений:

32. INSERT для отдельных столбцов, подобно привилегии UPDATE;
33. REFERENCES – для поддержки внешнего ключа.

Помимо перечисленных выше добавлена привилегия USAGE – для других объектов базы данных.

Кроме того, большинство коммерческих СУБД поддерживает дополнительные привилегии, например:

34. ALTER – позволяет модифицировать структуру таблиц (DB2, Oracle);

35. EXECUTE – позволяет выполнять хранимые процедуры.

Создатель объекта также получает право предоставить привилегии доступа какому-нибудь другому пользователю с помощью оператора GRANT. Ниже приводится синтаксис утверждения GRANT:

GRANT {SELECT|INSERT|DELETE|(UPDATE столбец, …)}, …

ON таблица TO {пользователь | PUBLIC} [WITH GRANT OPTION]

Привилегии вставки (INSERT) и обновления (UPDATE) (но не привилегии выбора SELECT, что весьма странно) могут задаваться для специально заданных столбцов.

Если задана директива WITH GRANT OPTION, это значит, что указанные пользователи наделены особыми полномочиями для заданного объекта – правом предоставления полномочий. Это, в свою очередь, означает, что для работы с данным объектом они могут наделять полномочиями других пользователей

Например: предоставить пользователю Ivanov полномочия для осуществления выборки и модификации фамилий в таблице Students с правом предоставления полномочий.

GRANT SELECT, UPDATE StName

ON Students TO Ivanov WITH GRANT OPTION

Если пользователь A наделяет некоторыми полномочиями другого пользователя B, то впоследствии он может отменить эти полномочия для пользователя B. Отмена полномочий выполняется с помощью директивы REVOKE с приведенным ниже синтаксисом.

REVOKE {{SELECT | INSERT | DELETE | UPDATE},…|ALL PRIVILEGES}

ON таблица,… FROM {пользователь | PUBLIC},… {CASCADE | RESTRICT}

Поскольку пользователь, с которого снимается привилегия, мог предоставить ее другому пользователю (если обладал правом предоставления полномочий), возможно возникновение ситуации покинутых привилегий. Основное предназначение параметров RESTRICT и CASCADE заключается в предотвращении ситуаций с возникновением покинутых привилегий. Благодаря заданию параметра RESTRICT не разрешается выполнять операцию отмены привилегии, если она приводит к появлению покинутой привилегии. Параметр CASCADE указывает на последовательную отмену всех привилегий, производных от данной.

Например: снять с пользователя Ivanov полномочия для осуществления модификации фамилий в таблице Students. Также снять эту привилегию со всех пользователей, которым она была предоставлена Ивановым.

REVOKE UPDATE

ON Students FROM Ivanov CASCADE

При удалении домена, таблицы, столбца или представления автоматически будут удалены также и все привилегии в отношении этих объектов со стороны всех пользователей.

Представления и безопасность

Создавая представления, и давая пользователям разрешение на доступ к нему, а не к исходной таблице, можно тем самым ограничить доступ пользователя, разрешив его только к заданным столбцам или записям. Таким образом, представления позволяют осуществить полный контроль над тем, какие данные доступны тому или иному пользователю.

Продемонстрируем использование представлений для обеспечения безопасности с помощью описанных на языке SQL примеров.

36. Создание представления для доступа к данным группы А–98–51.

```
CREATE VIEW GroupA98 AS
SELECT *
FROM Students INNER JOIN Groups ON Students.GrNo = Groups.GrNo
WHERE Groups.GrName = 'A-98-51'
```

37. Предоставление полномочий пользователю Ivanov.

```
GRANT SELECT, INSERT, DELETE, UPDATE
ON GroupA98 TO Ivanov
```

Теперь пользователь Ivanov может просматривать и модифицировать только данные группы А–98–51.

Тема 9. ФИЗИЧЕСКАЯ ОРГАНИЗАЦИЯ БД: СТРУКТУРЫ ХРАНЕНИЯ И МЕТОДЫ ДОСТУПА

В этой лекции рассматриваются технологии физического хранения данных хранящимся на диске и осуществления доступа к ним. (Далее под термином диск будут подразумеваться все устройства хранения данных прямого доступа, т.е. прежде всего традиционные жесткие диски с подвижными магнитными головками, внешние запоминающие устройства большой емкости, оптические диски и др.)

Внимание специалистов к структурам хранения и методам доступа вызвано очень медленной внешней памятью. Основным способом повышения производительности является минимизация числа дисковых операций ввода-вывода данных.

Как упоминалось ранее, любое упорядочение (расположение) данных на диске называется структурой хранения. Можно организовать самые разные структуры хранения, обладающие различной производительностью и оптимальные для различных способов использования. Однако не существует идеальной структуры хранения, которая была бы оптимальна для любых задач. Исходя из этого, можно заключить, что совершенная СУБД должна содержать несколько разных структур хранения для различных частей системы. Кроме того, следует также предусмотреть возможность изменения структуры хранения по мере изменения требований к производительности системы.

Работа СУБД построена следующим образом и включает три основных этапа (рис. 2).

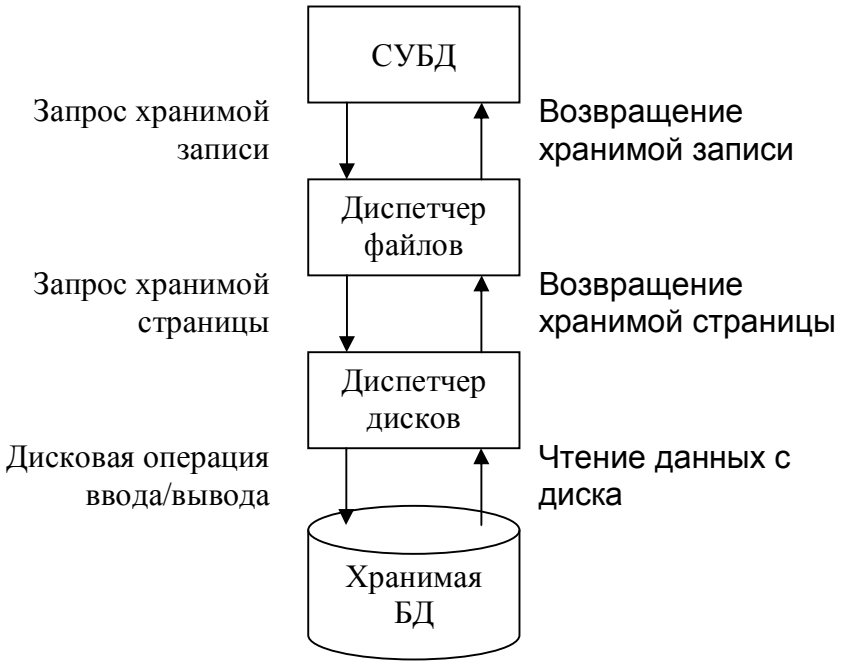

рис. 2 Схема взаимодействия СУБД, диспетчера файлов и диспетчера дисков.

1. Сначала в СУБД определяется искомая запись, а затем для ее извлечения запрашивается диспетчер файлов.
2. Диспетчер файлов определяет страницу, на которой находится искомая запись, а затем для извлечения этой страницы запрашивает диспетчер дисков.
3. Наконец, диспетчер дисков определяет физическое положение искомой страницы на диске и посылает соответствующий запрос на ввод-вывод данных.

Диспетчер дисков. Диспетчер дисков является компонентом используемой операционной системы, с помощью которого выполняются все дисковые операции ввода-вывода (в некоторых операционных системах он называется компонентом базовой системы ввода-вывода).

Для выполнения этих операций необходимо знать значения физических адресов на диске. Например, если диспетчер файлов запрашивает некоторую

страницу диспетчеру дисков необходимо знать, где конкретно находится страница на физическом диске. Однако "пользователю" диспетчера дисков, т.е. диспетчеру файлов, не обязательно знать физические адреса. Вместо этого диспетчеру файлов достаточно рассматривать диск как набор страниц фиксированного размера (т.е. набор "ячеек" памяти одинакового размера) с уникальным идентификационным номером набора страниц. Каждая страница, в свою очередь, обладает уникальным внутри данного набора идентификационным номером страницы, причем наборы не имеют общих страниц. Соответствие физических адресов на диске и номеров страниц достигается с помощью диспетчера дисков. Главным (и не единственным) преимуществом такой организации является изоляция программного кода, зависящего от конкретного устройства диска, внутри одного из компонентов системы, а именно внутри диспетчера дисков. В таком случае все компоненты высокого уровня, в частности диспетчера файлов, могут быть аппаратно независимыми.

Диспетчер файлов. При работе с диском как набором хранимых файлов, диспетчер файлов использует все имеющиеся средства диспетчера дисков согласно определенному в СУБД способу. При этом каждый набор страниц может содержать один или несколько хранимых файлов.

Каждый хранимый файл имеет имя (file name) или идентификационный номер (file ID), уникальные в данном наборе страниц. А каждая хранимая запись, в свою очередь, обладает идентификационным номером записи (record number или record ID), уникальным, по крайней мере, в пределах данного хранимого файла.

Кластеризация

Нельзя завершить этот краткий обзор без упоминания технологии кластеризации данных. В ее основе лежит принцип как можно более близкого физического размещения на диске логически связанных между собой и часто используемых данных. Физическая кластеризация данных – чрезвычайно важное условие высокой производительности, что можно продемонстрировать

следующим примером. Допустим, что наиболее часто используется хранимая запись r1 страницы p1, для работы с которой также требуется вызывать хранимую запись r2 страницы p2. Тогда возможно возникновение следующих ситуаций:

1. Если страницы p1 и p2 совпадают, то для доступа к записи r2 не потребуется выполнять еще одну физическую операцию ввода-вывода, поскольку нужная страница уже будет находиться в оперативной памяти.
2. Если страницы p1 и p2 не совпадают, но физически размещаются достаточно близко, например смежные страницы, то для доступа к записи r2 потребуется выполнить еще одну физическую операцию ввода-вывода (если, конечно, страница p2 еще не находится в оперативной памяти). Однако, поскольку головка чтения/записи уже будет находиться в непосредственной близости от нужного положения, время поиска будет очень малым. А если страницы p1 и p2 находятся на одном цилиндре, время поиска вообще будет равно нулю.

Внутрифайловую и межфайловую кластеризацию СУБД может осуществлять, размещая логически связанные записи на одной странице (если это возможно) или на смежных страницах (в противном случае).

Кластеризация внутри СУБД возможна только в том случае, если администратор базы данных организует ее. В совершенных СУБД часто предусмотрено задание нескольких различных типов кластеризации данных из разных файлов.

Индексирование

Рассмотрим в качестве примера таблицу с данными о студентах, а также часто используемый и потому очень важный запрос типа "Найти всех студентов учащихся в группе X", где X – некий параметр. При таких условиях администратор базы данных может выбрать способ сохранения данных, схематически показанный на рис. 3. Он основан на двух хранимых файлах: файле с данными о студентах и файле с данными о группах; файлы могут размещаться в различных наборах страниц. Предполагается, что в файле групп

используется упорядочение по алфавитному перечню их названий, т.е. по ключевому полю GrName (название группы) с указателями на соответствующие записи в файле поставщиков.

Файл студентов (данные)

StNo	GrName	StName	City
1	А–98–51	Иванов	Желтые Воды
4	Б–99–51	Стрельцов	Львов
2	А–98–51	Петров	Пятихатки
5	Б–99–51	Кузнецов	Львов
3	А–98–51	Сидоров	Пятихатки

Файл групп (индекс)

	А–98–51
	А–98–51
	А–98–51
	Б–99–51
	Б–99–51

рис. 3 Индексирование файла поставщиков по полю CITY файла городов.

Для поиска всех студентов из группы Б-99-51 можно применить следующую стратегию: найти в файле групп группу Б-99-51, а затем согласно указателям извлечь все соответствующие записи из файла студентов.

Такая стратегия будет более эффективной по сравнению с поиском в файле с данными студентов, поскольку, СУБД известна физическая последовательность записей в файле групп (поиск будет прекращен после извлечения следующей за Б-98-51 названия группы в алфавитном порядке). Кроме того, даже если придется просмотреть файл групп полностью, для такого поиска потребуется гораздо меньше операций ввода-вывода, поскольку физический размер файла групп меньше, чем размер файла с данными студентов из-за меньшего размера записей.

В рассматриваемом примере файл групп называется индексным файлом или индексом по отношению к файлу студентов, и наоборот, файл студентов индексирован (называется индексированным файлом) по отношению к файлу групп.

Индексный файл – это хранимый файл особого типа, в котором каждая запись состоит из двух значений, а именно данных и указателя. Данные соответствуют некоторому полю (индексному полю) из индексированного файла, а указатель служит для связывания с соответствующей записью индексированного файла. Индексное поле также называется индексным ключом (index key).

Индекс можно сравнить с предметным указателем обычной книги, который состоит из списка слов с "указателями" (номерами страниц) для упрощения поиска связанной с этими словами информации из "индексированного файла" (т.е. из содержимого книги).

Основным преимуществом использования индексов является значительное ускорение процесса выборки или извлечения данных, а основным недостатком – замедление процесса обновления данных, поскольку при каждом добавлении новой записи в индексированный файл потребуется также добавить новый индекс в индексный файл.

Хранимый файл может иметь несколько индексов, которые могут как раздельно, так и совместно использоваться для более эффективного доступа к записям о поставщиках.

Индексы часто называют инвертированными списками. Дело в том, что если файл студентов (см. рис. 3) имеет традиционную структуру списка набора значений полей для каждой записи, то индекс содержит список набора записей для каждого значения индексированного поля.

Индекс можно также создать на основе комбинации двух или более полей. Например, на рис. 4 показана схема индексирования файла студентов на основе комбинации полей GrName и City. При такой организации в СУБД можно выполнить запрос типа "Найти студентов учащихся в группе Б-98-51 проживающих в г. Кривой Рог" на основе однократного просмотра с помощью одного индекса.

Файл студентов (данные)

StNo	GrName	StName	City
1	А–98–51	Иванов	Желтые Воды
4	Б–99–51	Стрельцов	Львов
2	А–98–51	Петров	Пятихатки
5	Б–99–51	Кузнецов	Львов
3	А–98–51	Сидоров	Пятихатки

индекс GrName/City

	А–98–51/Желтые Воды
	А–98–51/Пятихатки
	А–98–51/Пятихатки
	Б–99–51/Львов
	Б–99–51/Львов

рис. 4 Индексирование файла поставщиков на основе комбинации полей GrName и City

Обратите внимание, что комбинированный индекс GrName/City может также служить индексом по одному полю GrName, поскольку все записи в комбинированном индексе расположены последовательно.

Плотное и неплотное индексирование

Основной целью использования индекса является ускорение процесса извлечения данных, точнее, уменьшение числа дисковых операций ввода-вывода, необходимых для извлечения требуемой записи. В основном это достигается благодаря использованию указателей. Хотя до сих пор предполагалось, что в этом качестве используются указатели записей, на самом деле для этого достаточно было бы указателей страниц (т.е. номеров страниц). Конечно, для последующего поиска записи внутри данной страницы придется осуществить еще одну операцию извлечения записи, однако теперь она будет выполняться в оперативной памяти и для этого не придется увеличивать число дисковых операций ввода-вывода.

Эту идею можно развить дальше, если вспомнить, что данные в каждом хранимом файле находятся в единой "физической" последовательности на основе комбинации последовательности хранимых записей внутри каждой страницы и последовательности страниц внутри каждого набора страниц. Предположим, что физическая последовательность файла студентов соответствует логической последовательности, заданной на основе некоторого поля, например номера студента. Иначе говоря, в этом файле выполнена кластеризация по данному полю. Допустим, что по этому же полю осуществляется индексирование; тогда нет необходимости в данном индексе хранить указатели для каждой записи индексируемого файла (в данном случае для файла студентов). Все, что требуется, – это указатель для каждой страницы, состоящий из максимального номера студента для данной страницы и соответствующего номера страницы. Схематически такая структура показана на рис.4, где для простоты предполагается, что на каждой странице может размещаться максимум две записи.

рис. 5 Рис. А. 12 Пример использования неплотного индекса.

В качестве примера рассмотрим процесс извлечения записи с номером 3 с помощью такого индекса. Сначала в СУБД проводится поиск индекса для записи с номером, большим или равным 3. При этом будет найдено поле с номером 4, которое содержит указатель на страницу p. Страница p извлекается, помещается в оперативную память и просматривается для поиска заданной хранимой записи (которая в данном примере будет найдена очень быстро).

Индекс с описанной структурой называется неплотным (или разряженным), поскольку в нем не содержатся указатели на все записи индексированного файла. Схематически пример такого индекса показан на рис.4и(Все описанные выше индексы, наоборот, называются плотными.) Одним из преимуществ неплотных индексов является их малый размер по сравнению с плотными индексами, так как они содержат меньшее число записей. Это часто позволяет просматривать содержимое базы данных с большей скоростью. Однако с помощью одного только неплотного индекса нельзя выполнить проверку наличия некоторого значения.

Следует отметить, что в данном хранимом файле может быть по крайней мере один неплотный индекс, который организуется на основе (уникальной) физической последовательности, заданной в файле. А все другие индексы обязательно должны быть плотными.

Структуры типа Б-дерева

Одним из наиболее важных и распространенных индексов является структура типа Б-дерева (B-tree).

Причина необходимости создания структуры типа Б-дерева заключается в желании избежать обязательного просмотра всего содержимого индексированного файла согласно его физической последовательности. Дело в том, что если индексированный файл имеет большой размер, то и его индекс также очень велик. Поэтому последовательный просмотр даже одного только индекса требует больших затрат времени. Разрешить эту проблему можно тем же способом, что и раньше: рассмотреть индексный файл как обычный хранимый файл и создать для него еще один индекс. Эту операцию можно осуществлять повторно нужное количество раз (обычно она применяется трижды, поскольку создание большого количества иерархических уровней индексирования требуется для очень больших файлов). При этом индекс на каждом из уровней будет неплотным по отношению к нижнему индексируемому уровню (он обязательно должен быть неплотным, иначе такая структура бессмысленна, так как уровень n содержал бы такое же количество записей, что и уровень n+1, а для просмотра потребовалось бы такое же длительное время).

Структура типа Б-дерева является частным случаем индекса древовидного типа и впервые описана в статье Байера (Bayer) и Мак-Крайта (McCreight) в 1972 году. С тех пор Байером и другими исследователями было предложено множество вариантов реализации этой идеи. В результате бинарные индексы различных типов стали широко использоваться во всех современных СУБД.

В варианте Кнута индекс состоит из двух частей:

3. Набор последовательностей включает одноуровневый индекс для реальных данных, который обычно является плотным, но может быть и неплотным, если в индексированном файле проведена кластеризация на основе индекса
4. Набор индексов, в свою очередь, обеспечивает быстрый непосредственный доступ к набору последовательностей (а значит, и к данным). По сути, набор индексов является древовидным индексным файлом для набора последовательностей или, строго говоря, индексом со структурой Б-дерева.

Комбинация набора индексов и набора последовательностей называется структурой типа Б-плюс-дерева (B-plus tree или B-tree). На рис. 6 показан простой пример такой структуры.

Числа 6, 8, 12, ... 97, 99 являются значениями индексированного поля F. Корневой элемент содержит два значения поля F (50 и 82) и три указателя (номера страниц). Данные со значением поля F, равным или меньшим 50, могут быть найдены с помощью левого указателя; данные со значением поля F, большим 50 и равным или меньшим 82, – с помощью среднего указателя; наконец, данные со значением поля F, большим 82, – с помощью правого указателя. Другие элементы набора индексов следует интерпретировать подобным образом. Обратите внимание, что благодаря переходу на второй уровень по левому указателю в дальнейшем поиск по правому указателю будет осуществляться ко всем записям со значением поля F, большим 32 и равным или меньшим 50.

Вообще, Б-дерево порядка п содержит не менее п и не более 2п записей с данными в каждом из элементов структуры (для каждых k записей требуется также k+1 указателей). Кроме того, ни одна из записей не может использоваться двумя разными элементами.

Одним из недостатков иерархических структур является несбалансированность их работы после удаления или вставки некоторых элементов. Дело в том, что в результате таких изменений структуры элементы с реальными данными могут оказаться на разных уровнях и на разных расстояниях от корневого элемента. Поскольку во время поиска при каждом посещении элементов структуры происходит обращение к диску, общая продолжительность поиска в несбалансированной древовидной структуре может оказаться совершенно непредсказуемой.

Преимуществом структуры типа Б-дерева является возможность сбалансированной вставки или удаления значений. (Вот почему для английского написания такого индекса, "B-tree", иногда употребляют вместо символа "B" эпитет от "сбалансированный" (balanced).) Ниже приводится

краткий алгоритм вставки нового значения V в структуру типа Б-дерева порядка п. Он рассчитан на вставку значения только лишь в набор индексов, но может быть достаточно просто расширен для вставки записи с данными в набор последовательностей.

5. На самом низком уровне набора индексов следует найти элемент (допустим, что это элемент N), с которым логически связано вставляемое значение V. Если элемент N содержит свободное пространство, то значение V вставляется в него и на этом процесс завершается.
6. В противном случае (если свободного пространства нет, т.е. придется создать еще один уровень) элемент N (допустим, что он содержит 2n индексных записей) разделяется на два элемента – N1 и N2. Обозначим символом 5 множество из 2n+1 значений, в котором 2n исходных значений и одно новое значение V. Тогда n первых значений этой логической (уже упорядоченной) последовательности необходимо поместить в элемент N1, n последних – в элемент N2, а среднее между ними значение W– в родительский элемент P на более высоком структурном уровне. Впоследствии, при осуществлении поиска значения U и достижении элемента P, поиск будет перенаправлен в сторону элемента N1, если V<W, либо в сторону элемента N2, если U> W.
7. Далее этот процесс следует повторить для вставки среднего значения W в родительский элемент P на более высоком структурном уровне.

В худшем случае процесс разделения элементов структуры может продлиться вплоть до корневого элемента всей структуры с образованием нового иерархического уровня.

Для удаления некоторого значения следует применить аналогичный алгоритм, но только в обратном порядке. А для изменения значения его можно удалить, а затем вставить новое.

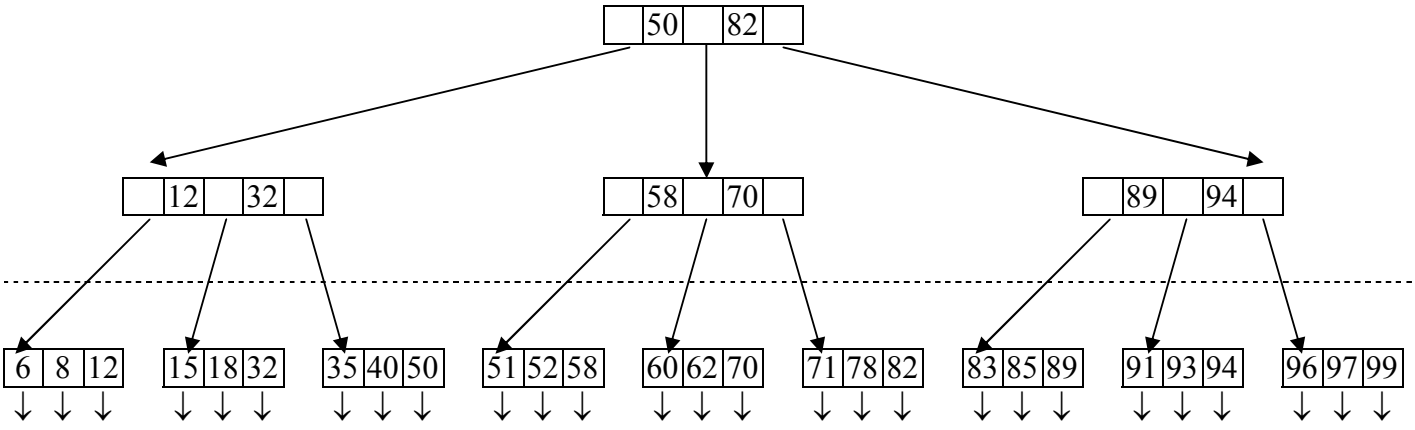

рис. 6 Пример структуры типа Б-дерева

Хеширование

Хешированием, хеш-адресацией или хеш-индексированием называется технология быстрого прямого доступа к хранимой записи на основе заданного значения некоторого поля. При этом не обязательно, чтобы поле было ключевым.

Ниже перечислены основные черты этой технологии:

8. каждая хранимая запись базы данных размещается по адресу, который вычисляется с помощью специальной хеш-функции на основе значения некоторого поля данной записи, т.е. хеш-поля, или хеш-ключа. Вычисленный адрес называется хеш-адресом.
9. для сохранения записи в СУБД сначала вычисляется хеш-адрес новой записи, а затем диспетчер файлов помещает эту запись по вычисленному адресу.
10. Для извлечения нужной записи по заданному значению хеш-поля в СУБД сначала вычисляется хеш-адрес, а затем диспетчеру файлов посылается запрос для извлечения записи по вычисленному адресу.

В качестве простой иллюстрации предположим, что у нас есть записи с данными о студентах с кодами 100, 200, 300, 400, 500, а в качестве хеш-функции h используется следующая: h = StNo mod 13, где h – хеш-адрес, StNo – код студента.

Это простейший пример общего класса хеш-функций типа деление/остаток. (В качестве делителя следует выбирать простое натуральное число). В этом примере хеш-адресами для заданных записей будут 9, 5, 1, 10 и 6 соответственно. Схематически взаимное расположение записей на страницах показано на

рис. **7**.

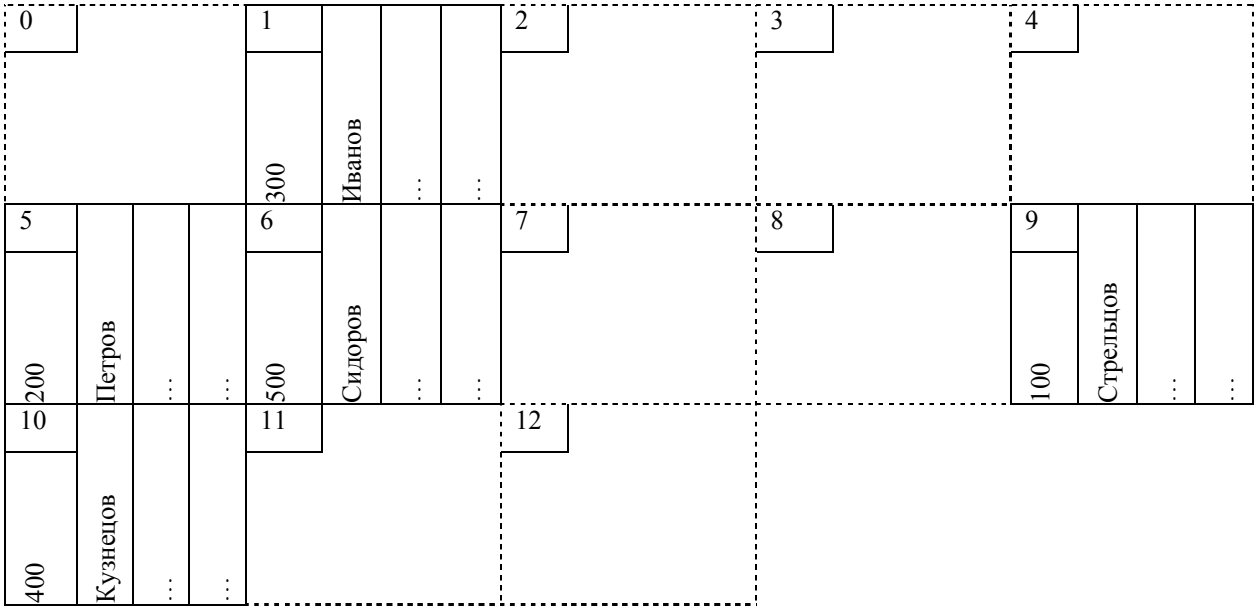

рис. 7 Пример использования хеширования.

Недостатком хеширования является возможность возникновения коллизий, т.е. ситуаций, когда две или более различных записи имеют одинаковые адреса. Допустим, что файл студентов из предыдущего примера (с записями 100, 200 и т.д.) содержит также запись с номером 1400. При использовании хеш-функции h = StNo mod 13 возникнет коллизия (по адресу 9) с записью 100.

Для разрешения таких коллизий можно использовать значения хеш-функции в качестве адреса не какой-либо записи с данными, а точки привязки. Точка привязки – это начальный адрес цепочки указателей (цепочки коллизий), связывающей вместе все записи или все страницы с записями, которые вызывают коллизии по адресу. Внутри цепочки коллизий записи также могут быть упорядочены согласно хеш-полю для упрощения последующего поиска.

Один из недостатков описанного выше способа хеширования – возрастание числа коллизий с увеличением размера хранимого файла. Это, в

свою очередь, может привести к значительному увеличению среднего времени доступа, поскольку все больше и больше времени придется тратить на поиск записей в наборах конфликтующих записей. Однако этот недостаток можно устранить, если реорганизовать файл, т.е. выгрузить данный файл и загрузить его вновь, используя новую хеш-функцию.

Существует ряд модификаций алгоритма хеширования, например, расширяемое хеширование и др., применяемые для оптимизации операций обновления и поиска в БД.

Тема 9. ОПТИМИЗАЦИЯ ЗАПРОСОВ

Для реляционных систем оптимизация является как проблемой, так и возможностью повышения производительности. Проблема оптимизации состоит в том, что некоторые системы для достижения определенного уровня производительности требуют оптимизации. Оптимизация позволяет улучшить работу системы, так как одной из сильных сторон реляционного подхода является то, что первое применение оптимизации к реляционному выражению переводит это выражение на более эффективный семантический уровень. Общее назначение оптимизатора состоит в выборе эффективной стратегии для вычисления данного реляционного выражения.

Преимущество автоматической оптимизации заключается в том, что пользователь может не задумываться над наилучшим способом выражения своих запросов (т.е. над тем, как сформулировать запрос, чтобы система выполнила его с максимально возможной производительностью). Но это далеко не все. Существует реальная возможность, что оптимизатор сформулирует запрос лучше, чем программист-пользователь. Для подобного утверждения есть ряд причин. Ниже приведены только некоторые из них:

4. Хороший оптимизатор – обратите внимание на слово "хороший" – обладает достаточным количеством информации, которой пользователь может не иметь. Точнее, оптимизатор должен обладать некоторыми статистическими данными, такими как кардинальное число каждого базового отношения, количество различающихся значений для каждого атрибута в отношении,

количество вхождений каждого значения в атрибутах и т.п. Благодаря наличию этих данных оптимизатор способен более точно оценивать эффективность любой стратегии реализации конкретного запроса. Поэтому оптимизатор сможет выбрать наилучшую стратегию реализации запроса.

5. Если с течением времени статистика базы данных значительно изменится (например, база данных будет физически реорганизована), то для реализации запроса может потребоваться совсем иная стратегия, чем до реорганизации. Другими словами, может понадобиться повторная оптимизация, или реоптимизация. В реляционных системах процесс реоптимизации достаточно тривиален – это просто повторная обработка исходного запроса системным оптимизатором. С другой стороны, в нереляционных системах реоптимизация требует переписывания программы, и, возможно, невыполнима вообще.

6. Оптимизатор – это программа, поэтому он более "настойчив" по сравнению с человеком. Оптимизатор вполне способен рассматривать буквально сотни различных стратегий реализации данного запроса, в то время как программист вряд ли изучает более трех-четырех стратегий (по крайней мере, достаточно глубоко).

7. В оптимизатор встроены знания и опыт "лучших из лучших" программистов, что делает эти знания и опыт доступными для всех. Естественно, в противном случае широкому кругу пользователей будет предоставлен явно недостаточный набор дешевых и неэффективных возможностей.

Пример оптимизации реляционного выражения

Начнем изложение с простого примера, дающего представление о результатах, которые можно получить с помощью оптимизации. Рассмотрим запрос "Получить список фамилий студентов, учащихся в группе А-98-51". Алгебраическая запись этого запроса такова:

((Students JOIN Groups) WHERE GrName = 'A-98-51') [StName]

Предположим, что база данных содержит информацию о 100 группах и 10000 студентов, только 30 из которых обучаются в группе A-98-51. В таком случае, если система будет вычислять выражение прямо (т.е. вообще без оптимизации), то последовательность выполняемых действий будет выглядеть так:

8. Соединение отношений Students и Groups (по атрибуту GrNo). На этом этапе считывается информация о 10000 студентов и 10000 раз считывается информация о 100 группах (один раз для каждого студента). После этого создается промежуточный результат, состоящий из 10000 соединенных кортежей.

9. Выборка кортежей с данными только о группе A-98-51 из результата, полученного на этапе 1. На этом этапе создается новое отношение, которое состоит из 30 кортежей.

10. Проекция результата, полученного на этапе 2, по атрибуту StName. На этом этапе создается требуемый результат, состоящий из 30 кортежей.

Показанная ниже процедура эквивалентна описанной в том смысле, что обязательно создаст тот же конечный результат, но более эффективным способом:

11. Выборка кортежей с данными только о группе A-98-51 из отношения Groups. На этом этапе выполняется чтение 100 кортежей и создается результат, состоящий только из 1 кортежа.

12. Соединение результата, полученного на этапе 1, с отношением Students (по атрибуту GrNo). На этом этапе выполняется считывание данных о 10000 студентов и 10000 раз считывается информация о группе A-98-51, полученная на 1 этапе. Результат содержит 30 кортежей.

13. Проецирование результата, полученного на этапе 2, по атрибуту StName (аналогично этапу 3 предыдущей последовательности действий). Требуемый результат содержит 30 кортежей.

Первая из показанных процедур выполняет в общем 1010000 операций ввода-вывода кортежа, в то время как вторая процедура выполняет только 20000 операции ввода-вывода. Следовательно, если принять "количество операции ввода-вывода кортежа" в качестве меры производительности, то вторая процедура в 50 раз эффективнее первой. (На практике мерой производительности служит количество операций ввода-вывода страницы, а не одного кортежа, но для данного примера эту поправку можно игнорировать.)

Обзор процесса оптимизации

Стадия 1. Преобразование запроса во внутреннюю форму

На этой стадии выполняется преобразование запроса в некоторое внутреннее представление, более удобное для машинных манипуляций. Это полностью исключает из рассмотрения конструкции внешнего уровня (такие как "игра слов" конкретного синтаксиса рассматриваемого языка запросов) и готовит почву для последующих стадий оптимизации.

Обычно внутреннее представление запросов является определенной модификацией абстрактного синтаксического дерева, или дерева запроса.

Например, на рисунке показано дерево рассматриваемого выше в этой главе запроса ("Получить список фамилий студентов, учащихся в группе А-98-51").

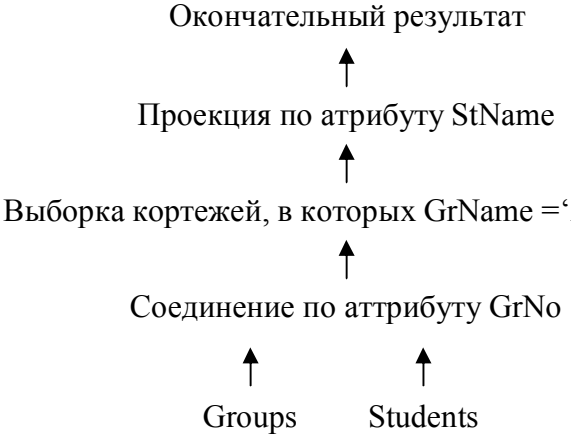

рис. 8. Дерево запроса "Получить список фамилий студентов, учащихся в группеА-98-51"

Стадия 2. Преобразование в каноническую форму

На этой стадии оптимизатор выполняет несколько операций оптимизации, которые "гарантированно являются хорошими" независимо от реальных данных, хранящихся в базе данных, и путей доступа к ним. Суть в том, что все запросы (за исключением простейших) реляционные языки обычно позволяют выразить несколькими разными (по крайней мере, внешне) способами.

Замечание о канонической форме. Понятие канонической формы употребляется, во многих разделах математики и связанных с ней дисциплин. Каноническая форма может быть определена следующим образом. Пусть Q – множество объектов (запросов), и пусть существует понятие об эквивалентности этих объектов (а именно: запросы q1 и q2 эквивалентны тогда и только тогда, когда дают идентичные результаты) Говорят, что подмножество C множества Q является подмножеством канонических форм для запросов из Q в смысле определенной выше эквивалентности тогда и только тогда, когда каждому объекту q из Q соответствует только один объект c из C. Тогда говорят, что объект c является канонической формой объекта q. Все "интересующие" свойства, которыми обладает объект q, также присущи и объекту c. Поэтому, чтобы доказать различные "интересующие" результаты, достаточно изучить менее мощное множество объектов C, а не более мощное множество Q.

Чтобы преобразовать результаты стадии 1 в некоторую эквивалентную, но более эффективную форму, оптимизатор использует определенные и хорошо известные правила преобразования, или законы.

Стадия 3. Выбор потенциальных низкоуровневых процедур

После преобразования внутренней формы запроса в **более** подходящую (каноническую) форму оптимизатор должен решить, как выполнять запрос, представленный в канонической форме. На этой стадии принимается во внимание наличие индексов и других путей доступа, распределение хранимых

значений данных, физическая кластеризация хранимых данных и т.п. Заметьте, что на стадиях 1 и 2 этим вопросам совсем не уделялось внимания

Для каждой низкоуровневой операции оптимизатор обладает набором низкоуровневых процедур реализации.

Замечание. С каждой процедурой также связана стоимостная формула, которая указывает "стоимость" выполнения процедуры (т.е. уровень требуемых затрат на ее выполнение). Обычно стоимость вычисляется в контексте операций ввода-вывода с диска, но некоторые системы учитывают также время использования процессора и другие факторы. Эти стоимостные формулы используются на стадии 4.

Следовательно, далее с помощью информации из каталога о состоянии базы данных (существующие индексы, кардинальные числа отношений и т.п.) и данных о зависимостях, описанных выше, оптимизатор выберет одну или несколько процедур-кандидатов для каждой низкоуровневой операции в запросе. Этот процесс обычно называют выбором пути доступа.

Стадия 4. Генерация планов вычисления запроса и выбор плана с наименьшей стоимостью

На последней стадии процесса оптимизации конструируются потенциальные планы запросов, после чего следует выбор лучшего (т.е. наименее дорогого) плана выполнения запроса. Каждый план выполнения строится как комбинация набора процедур реализации, при этом каждой низкоуровневой операции в запросе соответствует одна процедура.

Для выбора плана с наименьшей стоимостью необходим метод привязки стоимости к данному плану. В основном стоимость плана – это просто сумма стоимостей отдельных процедур, которые использованы для его выполнения. Таким образом, работа оптимизатора сводится к вычислению стоимостных формул для каждой такой процедуры. Проблема состоит в том, что стоимость выполнения процедуры зависит от размера отношения (или отношений), которое выбранная процедура обрабатывает.

Преобразование выражений

Выборки и проекции

14. Последовательность выборок данного отношения может быть преобразована в одну (объединенную операцией AND) выборку этого отношения. Например, выражение

(A WHERE выборка_1) WHERE выборка_2

эквивалентно выражению

A WHERE выборка_1 AND выборка_2

15. В последовательности проекций данного отношения можно игнорировать все проекции, кроме последней. Таким образом, выражение

(A [проекция_1]) [проекция_2]

эквивалентно выражению

A [Проекция_2]

Конечно, чтобы первое выражение имело смысл, каждый атрибут, используемый в проекции_2, должен присутствовать и в проекции_1.

16. Выборку проекции можно трансформировать в проекцию выборки. Например, выражение

(A [проекция]) WHERE выборка

эквивалентно выражению

(A WHERE выборка) [проекция]

Заметьте, что в основном всегда полезно выполнять операцию выборки перед операцией проекции, так как выборка приведет к уменьшению размера входных данных для операции проекции и, следовательно, к уменьшению количества данных, которые нужно сортировать для исключения дублирующихся записей в процессе вычисления проекции.

Распределительный закон

Говорят, что унарный оператор распределяется по бинарной операции O, если для всех A и B выполняется условие

F (A O B) ≡ f (A) O f (B).

В реляционной алгебре операция выборки распределяется по операциям объединения, пересечения и вычитания. Операция выборки также распределяется по операции соединения, но только тогда, когда условие выборки состоит (в самом сложном случае) из объединенных операцией AND двух отдельных условий выборки – по одному для каждого операнда операции соединения. Для рассматриваемого выше в этой главе примера сформулированное условие соблюдено (условие выборки очень простое и относится лишь к одному операнду), и можно использовать распределительный закон для замены рассматриваемого в примере выражения его более эффективным эквивалентом. Чистый эффект этого закона состоит в том, что можно выполнять "раннюю выборку". Выполнение ранней выборки почти всегда себя оправдывает, так как приводит к значительному уменьшению количества кортежей, которые нужно рассматривать в следующей операции. Кроме того, ранняя выборка может привести к уменьшению количества кортежей и на выходе следующей операции.

Далее приведено несколько более специфических примеров распределительного закона, на этот раз с операцией проекции. Во-первых, операция проекции распределяется по операциям объединения и пересечения (но не по операции вычитания). Во-вторых, эта операция также распределяется по операции соединения, но только в том случае, если в проекцию включены все атрибуты соединения. Точнее, выражение

(A JOIN B) [проекция]

эквивалентно выражению

(A [A_проекция]) JOIN (B [B_проекция])

тогда и только тогда, когда множество использованных в проекции атрибутов равняется объединению множеств атрибутов в A_проекции и B_проекции и включает атрибуты, по которым выполнено соединение. Этот закон можно использовать для выполнения ранних "проекций", которые обычно себя оправдывают по тем же причинам, что и операции выборки.

Коммутативность и ассоциативность

Законы коммутативности и ассоциативности – это еще два общих правила преобразования. Говорят, что бинарная операция О является коммутативной, если для всех А и В истинно равенство

$$A \; O \; B \equiv B \; O \; A$$

Например, в обычной арифметике операции умножения и сложения являются коммутативными, а операции деления и вычитания – нет. В реляционной алгебре коммутативными являются операции объединения, пересечения и соединения, а операции вычитания и деления таковыми не являются.

Перейдем к ассоциативности. Принято считать, что бинарная операция О является ассоциативной, если для всех А, В и С истинно равенство

$$A \; O \; (B \; O \; C) \equiv (A \; O \; B) \; O \; C.$$

Например, в обычной арифметике произведение и сложение – ассоциативные операции, деление и вычитание – нет. В реляционной алгебре ассоциативными являются операции объединения, пересечения и соединения, а операции вычитания и деления таковыми не являются. Так, например, если в запросе используется соединение трех отношений, А, В и С, то из законов коммутативности и ассоциативности

Идемпотентность

Еще одним важным правилом является закон идемпотентности. Идемпотентной называют такую бинарную операцию О, для которой для всех А выполняется равенство

$$A \; O \; A = A.$$

Можно ожидать, что свойство идемпотентности также может быть полезным в процессе трансформации выражений. В реляционной алгебре операции объединения, пересечения и соединения являются идемпотентными, а операции деления и вычитания – нет.

Вычисляемые скалярные выражения

Предметом применения законов трансформации являются не только реляционные выражения. Например, уже было показано, что некоторые законы трансформации применимы и к арифметическим выражениям. Ниже приведен пример. Выражение

A * B + A * C

можно трансформировать в выражение

A * (B + C)

вследствие того, что операция умножения "*" распределяется по операции сложения "+". Оптимизатор реляционных выражений должен обладать информацией о подобных преобразованиях, так как он учитывает вычисляемые скалярные выражения в контексте операций EXTEND и SUMMARIZE.

Говорят, что бинарная операция O распределяется по бинарной операции O, если для всех A, B и C истинно равенство

A Ú (B O C) = (A Ú B) O (A Ú C)

(для приведенного выше арифметического примера замените Ú на "*", а O на "+").

Условия

Перейдем к обсуждению условий или выражений, результатами которых могут быть истина или ложь. Предположим, что A и B – атрибуты двух различных отношений, тогда условие

A>B AND B>3

(которое может быть частью запроса) абсолютно эквивалентно выражению

A > B AND B > 3 AND A > 3

и потому может быть преобразовано в это выражение.

Данная эквивалентность базируется на том, что операция ">" является транзитивной. Заметьте, что выполнение подобного преобразования весьма полезно, так как позволяет системе создать дополнительную выборку (с

помощью условия "A > 3") перед выполнением соединения "больше чем", требуемого условием "A > B".

Замечание. Этот прием реализован в различных коммерческих продуктах, включая систему DB2, в которой его называют транзитивным замыканием предикатов. А вот другой пример. Условие

A > B OR (C = D AND E < F)

можно преобразовать в условие

(A > B OR C = D) AND (A > B OR E < F)

вследствие того, что операция OR распределяется по операции AND. Этот пример демонстрирует другой общий закон: "Любое условие может быть преобразовано в эквивалентное условие, называемое конъюнктивной нормальной формой (КНФ)". КНФ-выражение имеет вид:

C1 AND C2 AND … AND Cn,

где C1, C2, ..., Cn – условия (называемые частичная конъюнкция), в которых не используется операция AND. Преимущество КНФ состоит в том, что КНФ-выражение истинно, только если истинны все его частичные конъюнкции. Аналогично, КНФ-выражение ложно, если ложь является результатом хотя бы одной частичной конъюнкции. Так как операция AND коммутативна (A AND B равно B AND A), то оптимизатор может вычислять отдельные частичные конъюнкции в любом порядке, в частности по возрастанию сложности (сначала простые). И как только найдена частичная конъюнкция, результатом которой является ложь, весь процесс вычисления КНФ-выражения можно останавливать.

Более того, в среде параллельных вычислений возможно параллельное вычисление всех частичных конъюнкций. Опять же, как только найдена первая частичная конъюнкция, результатом которой является ложь, весь процесс вычисления КНФ-выражения можно останавливать.

Семантические преобразования

Рассмотрим следующее выражение:

(Students JOIN Groups) [StName]

Данное соединение относится к соединениям типа внешниий-к-согласованному-потенциалъному-ключу. В этом соединении внешнему ключу в отношении Students ставится в соответствие потенциальный ключ отношения Groups. Следовательно, кортеж в отношении Students связан с определенным кортежем в отношении Groups. Таким образом, из каждого кортежа в отношении Students в общий результат поступает только значение атрибута StName. Другими словами, соединение можно не выполнять! Рассматриваемое выражение можно заменить выражением

Students [StName]

Преобразование, корректное в силу определенного условия целостности, называют семантическим преобразованием, а оптимизацию, полученную в результате подобных преобразований, – семантической оптимизацией. Семантическую оптимизацию можно определить как процесс преобразования запроса в другой, качественно отличный запрос, который, тем не менее, дает результат, идентичный результату исходного запроса, благодаря тому что данные удовлетворяют определенному условию целостности.

Важно понимать, что в принципе любое условие целостности может быть использовано для семантической оптимизации (если это условие не отсрочено и в данный момент действует на базу данных).

Статистики базы данных

На стадиях 3 и 4 общего процесса оптимизации (они называются стадиями "выбора пути доступа") используются так называемые статистики базы данных, которые хранятся в каталоге.

Тема 10. ВОССТАНОВЛЕНИЕ ПОСЛЕ СБОЕВ

Восстановление в системе управления базами данных, означает в первую очередь восстановление самой базы данных, т.е. возвращение базы данных в правильное состояние, если какой-либо сбой сделал текущее состояние неправильным или подозрительным. Основной принцип, на котором строится

такое восстановление, – это избыточность. Избыточность организуется на физическом уровне. Такая избыточность будет скрыта от пользователя, а следовательно, не видна на логическом уровне. Другими словами, если любая часть информации, содержащаяся в базе данных, может быть реконструирована из другой хранимой в системе избыточной информации, значит, база данных восстанавливаема.

Понятие транзакции

Транзакция – это логическая единица работы. Например. Предположим сначала, что отношение Students (отношение студентов) включает дополнительный атрибут AvgMark, представляющий собой средний балл студента, по результатам сдачи текущей сессии. Значение AvgMark для любой определенной детали предполагается равным среднему арифметическому всех значений Mark из таблицы Marks для всех оценок полученных в текущем семестре.

В приведенном примере предполагается, что речь идет об одиночной, атомарной операции. На самом деле добавление новой оценки в таблицу Marks – это выполнение двух обновлений в базе данных (под обновлениями здесь, конечно, понимаются операции insert, delete, а также сами по себе операции update). Более того, в базе данных между этими двумя обновлениями временно нарушается требование, что значение AvgMark для студента 1 равно среднему арифметическому всех значений поля Mark для студента 1 в текущем семестре. Таким образом, логическая единица работы (т.е. транзакция) – не просто одиночная операция системы баз данных, а скорее согласование нескольких таких операций. В общем, это преобразование одного согласованного состояния базы данных в другое, причем в промежуточных точках база данных находится в несогласованном состоянии.

Из этого следует, что недопустимо, чтобы одно из обновлений было выполнено, а другое нет, так как база данных останется в несогласованном состоянии. В идеальном случае должны быть выполнены оба обновления. Однако нельзя обеспечить стопроцентную гарантию, что так и будет. Не

исключена вероятность того, что, система, например, будет разрушена между двумя обновлениями, или же на втором обновлении произойдет арифметическое переполнение и т.п. Система, поддерживающая транзакции, гарантирует, что если во время выполнения неких обновлений произошла ошибка (по любой причине), то все эти обновления будут аннулированы. Таким образом, транзакция или выполняется полностью, или полностью отменяется (как будто она вообще не выполнялась).

Системный компонент, обеспечивающий атомарность (или ее подобие), называется администратором транзакций (или диспетчером транзакций), а ключами к его выполнению служат операторы `COMMIT TRANSACTION` и `ROLLBACK TRANSACTION`.

Оператор `COMMIT TRANSACTION` (для краткости commit) сигнализирует об успешном окончании транзакции. Он сообщает администратору транзакций, что логическая единица работы завершена успешно, база данных вновь находится (или будет находиться) в согласованном состоянии, а все обновления, выполненные логической единицей работы, теперь могут быть зафиксированы, т.е. стать постоянными.

Оператор `ROLLBACK TRANSACTION` (для краткости `ROLLBACK`) сигнализирует о неудачном окончании транзакции. Он сообщает администратору транзакций, что произошла какая-то ошибка, база данных находится в несогласованном состоянии и все обновления могут быть отменены, т.е. аннулированы.

Для отмены обновлений система поддерживает файл регистрации, или журнал, на диске, где записываются детали всех операций обновления, в частности новое и старое значения модифицированного объекта. Таким образом, при необходимости отмены некоторого обновления система может использовать соответствующий файл регистрации для возвращения объекта в первоначальное состояние.

Еще один важный момент. Система должна гарантировать, что индивидуальные операторы сами по себе атомарные (т.е. выполняются

полностью или не выполняются совсем). Это особенно важно для реляционных систем, в которых операторы многоуровневые и обычно оперируют множеством кортежей одновременно; такой оператор просто не может быть нарушен посреди операции и привести систему в несогласованное состояние. Другими словами, если произошла ошибка во время работы такого оператора, база данных должна остаться полностью неизмененной. Более того, это должно быть справедливо даже в том случае, когда действия оператора являются причиной дополнительной, например каскадной, операции.

Восстановление транзакции.

Транзакция начинается с успешного выполнения оператора `BEGIN TRANSACTION`) и заканчивается успешным выполнением либо оператора `COMMIT`, либо `ROLLBACK`. Оператор `COMMIT` устанавливает так называемую точку фиксации (которая в коммерческих продуктах также называется точкой синхронизации (syncpoint). Точка фиксации соответствует концу логической единицы работы и, следовательно, точке, в которой база данных находится (или будет находиться) в состоянии согласованности. В противовес этому, выполнение оператора `ROLLBACK` вновь возвращает базу данных в состояние, в котором она была во время операции `BEGIN TRANSACTION`, т.е. в предыдущую точку фиксации.

Случаи установки точки фиксации:

17. Все обновления, совершенные программой с тех пор, как установлена предыдущая точка фиксации, выполнены, т.е. стали постоянными. Во время выполнения все такие обновления могут расцениваться только как пробные (в том смысле, что они могут быть не выполнены, например прокручены назад). Гарантируется, что однажды зафиксированное обновление так и останется зафиксированным (это и есть определение понятия "зафиксировано").

18. Все позиционирование базы данных утеряно, и все блокировки кортежей реализованы. Позиционирование базы данных здесь означает, что в

любое конкретное время программа обычно адресована определенным кортежам. Эта адресуемость в точке фиксации теряется.

Следовательно, система может выполнить откат транзакции как явно – например по команде ПО с которым работает пользователь, так и неявно – для любой программы, которая по какой-либо причине не достигла запланированного завершения операций, входящих в транзакцию.

Из этого видно, что транзакции — это не только логические единицы работы, но также и единицы восстановления при неудачном выполнении операций. При успешном завершении транзакции система гарантирует, что обновления постоянно установлены в базе данных, даже если система потерпит крах в следующий момент. Возможно, что в системе произойдет сбой после успешного выполнения `COMMIT`, но перед тем, как, обновления будут физически записаны в базу данных (они все еще могут оставаться в буфере оперативной памяти и таким образом могут быть утеряны в момент сбоя системы). Даже если подобное случилось, процедура перезагрузки системы все равно должна устанавливать эти обновления в базу данных, исследуя соответствующие записи в файле регистрации. Из этого следует, что файл регистрации должен быть физически записан перед завершением операции `COMMIT`. Это важное правило ведения файла регистрации известно как протокол предварительной записи в журнал (т.е. запись об операции осуществляется перед ее выполнением). Таким образом, процедура перезагрузки сможет восстановить любые успешно завершенные транзакции, хотя их обновления не были записаны физически до аварийного отказа системы. Следовательно, как указывалось ранее, транзакция действительно является единицей восстановления.

Свойства АСИД.

Из предыдущих разделов следует, что транзакции обладают четырьмя важными свойствами: атомарность, согласованность, изоляция и долговечность (назовем это свойствами АСИД).

19. Атомарность. Транзакции атомарны (выполняется все или ничего).

20. Согласованность. Транзакции защищают базу данных согласованно. Это означает, что транзакции переводят одно согласованное состояние базы данных в другое без обязательной поддержки согласованности во всех промежуточных точках.

21. Изоляция. Транзакции отделены одна от другой. Это означает, что, если даже будет запущено множество конкурирующих друг с другом транзакций, любое обновление определенной транзакции будет скрыто от остальных до тех пор, пока эта транзакция выполняется. Другими словами, для любых двух отдаленных транзакций T1 и T2 справедливо следующее утверждение: T1 сможет увидеть обновление T2 только после выполнения T2, а T2 сможет увидеть обновление T1 только после выполнения T1.

22. Долговечность. Когда транзакция выполнена, ее обновления сохраняются, даже если в следующий момент произойдет сбой системы.

Алгоритм восстановления после сбоя системы

Система должна быть готова к восстановлению не только после небольших локальных нарушений, таких как невыполнение операции в пределах определенной транзакции, но также и после глобальных нарушений типа сбоев в питании вычислительного устройства и др. Местное нарушение по определению поражает только транзакцию, в которой оно собственно и произошло. Глобальное нарушение поражает сразу все транзакции и, следовательно, приводит к значительным для системы последствиям.

Существует два вида глобальных нарушений:

23. Отказы системы (например, сбои в питании), поражающие все выполняющиеся в данный момент транзакции, но физически не нарушающие базу данных в целом. Такие нарушения в системе также называют аварийным отказом программного обеспечения.

24. Отказы носителей (например, поломка головок дискового накопителя), которые могут представлять угрозу для базы данных или для какой-либо ее части и поражать. по крайней мере, те транзакции, которые

используют эту часть базы данных. Отказы носителей также называют аварийным отказом аппаратуры.

Восстановление после отказов системы

Критической точкой в отказе системы является потеря содержимого оперативной памяти (в частности, рабочих буферов базы данных). Поскольку точное состояние какой-либо выполняющейся в момент нарушения транзакции не известно, транзакция может не завершиться успешно и, таким образом, будет отменена при перезагрузке системы.

Более того, возможно, потребуется повторно выполнить определенную успешно завершившуюся до аварийного отказа транзакцию при перезагрузке системы, если не были физически выполнены обновления этой транзакции.

Для определения во время перезагрузки, какую транзакцию отменить, а какую выполнить повторно система в некотором предписанном интервале (когда в журнале накапливается определенное число записей) автоматически принимает контрольную точку. Принятие контрольной точки включает физическую запись содержимого рабочих буферов базы данных непосредственно в базу данных и специальную физическую запись контрольной точки, которая предоставляет список всех осуществляемых в данный момент транзакций. На рис. 9 рассматривается пять возможных вариантов выполнения транзакций до аварийного сбоя системы.

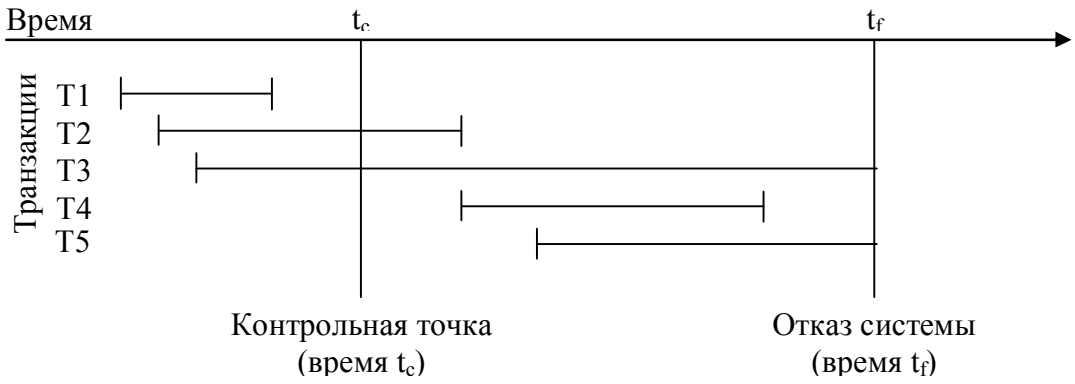

рис. 9 Варианты выполнения пяти транзакций.

Пояснения к рис. 9:

25. Отказ системы произошел в момент времени tf.

26. Близлежащая к моменту времени tf контрольная точка была принята в момент времени tc.

27. Транзакция T1 успешно завершена до момента времени tc.

28. Транзакция T2 начата до момента времени tc и успешно завершена после момента времени tc, но до момента времени tf.

29. Транзакция T3 также начата до момента времени tc, но не завершена к моменту времени tf

30. Транзакция T4 начата после момента времени tc и успешно завершена до момента времени tf.

31. Транзакция T5 также начата после момента времени tc, но не завершена к моменту времени tf.

Очевидно, что при перезагрузке системы транзакции типа T3 и T5 должны быть отменены, а транзакции типа T2 и T4 – выполнены повторно. Тем не менее заметьте, что транзакции типа T1 вообще не включаются в процесс перезагрузки, так как обновления попали в базу данных еще до момента времени tc (т.е. зафиксированы еще до принятия контрольной точки). Отметьте также, что транзакции, завершившиеся неудачно (в том числе отмененные) перед моментом времени tf, вообще не будут вовлечены в процесс перезагрузки.

Параллелизм. Проблемы параллелизма

Термин параллелизм означает возможность одновременной обработки в СУБД многих транзакций с доступом к одним и тем же данным, причем в одно и то же время. В такой системе для корректной обработки параллельных транзакций без возникновения конфликтных ситуаций необходимо использовать некоторый метод управления параллелизмом.

Каждый метод управления параллелизмом предназначен для решения некоторой конкретной задачи. Тем не менее, при обработке правильно составленных транзакций возникают ситуации, которые могут привести к получению неправильного результата из-за взаимных помех среди некоторых транзакций. (Обратите внимание, что вносящая помеху транзакция сама по себе

может быть правильной. Неправильный конечный результат возникает по причине бесконтрольного чередования операций из двух правильных транзакций). Основные проблемы, возникающие при параллельной обработке транзакций следующие:

32. проблема потери результатов обновления;
33. проблема незафиксированной зависимости;
34. проблема несовместимого анализа.

Проблема потери результатов обновления

Рассмотрим ситуацию, показанную на
рис. **10**, в такой интерпретации: транзакция A извлекает некоторый кортеж p в момент времени t1; транзакция B извлекает некоторый кортеж p в момент времени t2; транзакция A обновляет некоторый кортеж p (на основе значений, полученных в момент времени t1) в момент времени t3; транзакция B обновляет тот же кортеж p (на основе значений, полученных в момент времени t2, которые имеют те же значения, что и в момент времени t1) в момент времени t4. Однако результат операции обновления, выполненной транзакцией A, будет утерян, поскольку в момент времени t4 она не будет учтена и потому будет "отменена" операцией обновления, выполненной транзакцией B.

Транзакция A	Время	Транзакция B
Извлечение кортежа p	t1	–
–	t2	Извлечение кортежа p
Обновление кортежа p	t3	–
–	t4	Обновление кортежа p

рис. 10. Потеря в момент времени t4 результатов обновления, выполненного транзакцией A.

Проблема незафиксированной зависимости

Проблема незафиксированной зависимости появляется, если с помощью некоторой транзакции осуществляется извлечение (или, что еще хуже, обновление) некоторого кортежа, который в данный момент обновляется другой транзакцией, но это обновление еще не закончено. Таким образом, если обновление не завершено, существует некоторая вероятность того, что оно не

будет завершено никогда. (Более того, в подобном случае может быть выполнен возврат к предыдущему состоянию кортежа с отменой выполнения транзакции.) В таком случае, в первой транзакции будут принимать участие данные, которых больше не существует. Эта ситуация показана на рис. 11,

рис. **12**.

В первом примере (рис. 11) транзакция А в момент времени t2 встречается с невыполненным обновлением (оно также называется невыполненным изменением). Затем это обновление отменяется в момент времени t3. Таким образом, транзакция А выполняется на основе фальшивого предположения, что кортеж p имеет некоторое значение в момент времени t2, тогда как на самом деле он имеет некоторое значение, существовавшее еще в момент времени t1. В итоге после выполнения транзакции А будет получен неверный результат. Кроме того, обратите внимание, что отмена выполнения транзакции В может произойти не по вине транзакции В, а, например, в результате краха системы. (К этому времени выполнение транзакции А может быть уже завершено, а потому крушение системы не приведет к отмене выполнения транзакции А.)

Транзакция А	Время	Транзакция В
–	t1	Обновление кортежа p
Извлечение кортежа p	t2	–
–	t3	Отмена выполнения транзакции

рис. 11. Транзакция А становится зависимой от невыполненного изменения в момент времени t2.

Транзакция А	Время	Транзакция В
–	t1	Обновление кортежа p
Обновление кортежа p	t2	–
–	t3	Отмена выполнения транзакции

рис. 12. Транзакция А обновляет невыполненное изменение в момент времени t2, и результаты этого обновления утрачиваются в момент времени t3.

Второй пример, приведенный на

рис. **12**, иллюстрирует другой случай. Не только транзакция A становится зависимой от изменения, не выполненного в момент времени t2, но также в момент времени t3 фактически утрачивается результат обновления, поскольку отмена выполнения транзакции B в момент времени t3 приводит к восстановлению кортежа p к исходному значению в момент времени t1. Это еще один вариант проблемы потери результатов обновления.

Уровни изоляции транзакции

Термин уровень изоляции, грубо говоря, используется для описания степени вмешательства параллельных транзакций в работу некоторой заданной транзакции. Но при обеспечении возможности упорядочения не допускается никакого вмешательства, иначе говоря, уровень изоляции должен быть максимальным. Однако, как уже отмечалось, в реальных системах по различным причинам обычно допускаются транзакции, которые работают на уровне изоляции ниже максимального.

Уровень изоляции обычно рассматривается как некоторое свойство транзакции. В реальных СУБД может быть реализовано различное количество уровней изоляции.

Кроме того помимо кортежей могут блокироваться другие единицы данных, например целое отношение, база данных или (пример противоположного характера) некоторое значение атрибута внутри заданного кортежа.

Поддержка в языке SQL

SQL поддерживает операции `COMMIT` и `ROLLBACK` для фиксации и отката транзакции соответственно.

Специальный оператор `SET TRANSACTION` используется для определения некоторых характеристик транзакции, которую нужно будет инициировать, такие, как режим доступа и уровень изоляции.

В стандарте языка SQL не предусмотрена поддержка явным образом возможности блокировки (фактически, блокировка в нем вообще не упоминается). Блокировки накладываются неявно, при выполнении операторов SQL.